University of Alberta

Temporal-Difference Networks

by

Brian Timothy Tanner

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Master of Science**.

Department of Computing Science

Edmonton, Alberta

Fall 2005

Library and Archives Canada

Published Heritage Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et Archives Canada

Direction du Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 0-494-09297-1
Our file Notre référence
ISBN: 0-494-09297-1

NOTICE:
The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:
L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

Abstract

We introduce a generalization of temporal-difference (TD) learning to networks of interrelated predictions. Rather than relating a single prediction to itself at a later time, as in conventional TD methods, a TD network relates each prediction in a set of predictions to other predictions in the set at a later time. TD networks apply TD learning to a wider class of predictions than previously considered. We demonstrate that TD networks can learn predictive state representations that enable exact solution of non-Markov problems. We introduce two extensions to TD networks including augmenting the input representation to include recent history and generalizing the learning algorithm to use TD(λ) style eligibility traces. We argue that TD networks represent an extension to TD methods and bring us closer to the goal of representing world knowledge in predictive, grounded terms.

Acknowledgements

I would like to thank my supervisor Dr. Richard Sutton, for introducing me to an interesting area of research so early in my academic career. Also, for believing that I was worth the countless hours he spent helping improve my writing and research skills. I'd also like to thank everyone in the RLAI lab, but specially Eddie Rafols, Anna Koop, Dan Lizotte, Peter McCracken, David Silver, and Mark Ring. I would have been lost without all of those afternoon whiteboard sessions :)

Finally, I just want to thank my beautiful wife Ariel for having the patience to not only survive, but to support the late nights and the long hours. Thanks for understanding.

Contents

1 Introduction **1**

1.1 Discrete Dynamical Systems 1

1.2 Predictive Representations 2

1.3 Temporal-Difference Networks 3

1.4 TD Network Overview 4

1.4.1 Question Networks 4

1.4.2 TD Network Operation 6

2 Related Work **10**

2.1 Temporal-Difference Methods of Prediction 10

2.2 Discrete Dynamical Systems 11

2.3 Modeling Dynamical Systems 12

2.3.1 Markov Decision Processes 14

2.3.2 N^{th}-Order Markov Models 15

2.3.3 The System-Dynamics Matrix 15

2.3.4 Partially Observable Markov Decision Processes 18

2.3.5 Predictive Representations 20

2.3.6 Linear Predictive State Representations 20

2.3.7 TD Networks and Predictive State Representations 21
2.3.8 Nonlinear Predictive State Representations 24
2.3.9 Schemas 25
2.3.10 Variable Length Memory Methods 25
2.3.11 Other Methods 26
2.4 Artificial Neural Networks 26
2.4.1 TD Networks and Artificial Neural Networks 27

3 TD Network Experiments **31**
3.1 Error Reporting 31
3.2 Random Walk 32
3.2.1 Experiment 1: n-step Unconditional Prediction 33
3.2.2 Experiment 2: Action-conditional Prediction 36
3.2.3 Experiment 3: Learning a Predictive State Representation . . 38
3.3 Conclusion 40

4 TD Networks with History **41**
4.1 TD Network Counter-examples 41
4.1.1 Experimental Results 44
4.1.2 Approximate Solutions 46
4.2 Indefinite-memory Problems 47
4.2.1 Experimental Results 47
4.3 Conclusions 49

5 TD(λ) Networks **50**
5.1 Questions and Targets 51

5.2 TD(λ) Networks 52

5.2.1 TD(λ) Network Learning Algorithm 55

5.2.2 TD(λ) Network Learning Algorithm Discussion 56

5.3 Experimental Results 57

5.4 The Computational Cost of λ 62

5.5 Conclusions 65

6 Conclusion **67**

6.1 Discussion and Contributions 67

6.2 Future Work 68

6.2.1 Discovery 69

6.2.2 Fast Learning 69

6.2.3 Temporal Abstraction 70

6.2.4 Reinforcement Learning 70

Bibliography **71**

List of Figures

1.1 Three sample question networks. 5

2.1 Simple MDP with 5 states and 2 actions. 14

2.2 8-state ring world 19

2.3 Artificial neural network 27

2.4 TD network drawn as a recurrent neural network 28

2.5 Jordan's recurrent ANN 29

2.6 Elman's recurrent ANN 29

3.1 7-state stochastic random walk world. 33

3.2 Fully observable 7-state stochastic random walk world 34

3.3 Sample question networks 35

3.4 RMSE of TD-network predictions on non-Markov task 39

4.1 4-state cycle world 42

4.2 Sample input vector for cycle world 44

4.3 TD network performance on 7-state cycle world 45

4.4 5-state ring world and question network 48

4.5 Performance of TD network on 5-state ring world 49

5.1 Symmetric action-conditional question network. 51

5.2 Target flow diagram 53

5.3 Pseudo-code for TD(λ) learning algorithm. 55

5.4 Learning curves for TD(λ) learning algorithm on cycle world 58

5.5 8-state ring world and question network 59

5.6 Learning curves for TD(λ) learning algorithm on 5-state ring world . 60

5.7 Learning curves for TD(λ) learning algorithm on 8-state ring world . 61

List of Tables

2.1 Observation matrix : 2 observations and actions. 13

2.2 Observation matrix : MDP. 15

2.3 $\mathcal{P}$ function : 2^{nd}-order Markov model. 16

2.4 Systems-dynamics matrix: 2 observations and actions 18

3.1 RMSE of Monte-Carlo and TD-network predictions (no actions) . . 36

3.2 RMSE of Monte-Carlo and TD-network predictions (with actions) . 37

3.3 Batch performanec of Monte-Carlo and TD-network predictions . . . 38

4.1 Unstable approximate solutions learned by a TD network 46

5.1 RMSE of EM and TD(1) algorithm on cycle and ring worlds 61

Chapter 1

Introduction

Predicting future experience from past experience is a problem of interest in a variety of disciplines including psychology, economics, engineering, and physics. Whether the objective is to estimate the course of a distant comet or a stock's opening price, prediction requires a *model*: a simplified description of a complex process.

A *paradigm* or *representation* is the set of beliefs and assumptions that provide the foundation for a particular class of models. Each paradigm has its own own strengths and weaknesses. For example, vast amounts of computation may be required to create a theoretically correct model in one paradigm, while another representation may be used to produce inaccurate models inexpensively.

This thesis explores a new predictive knowledge representation, called temporal-difference (TD) networks.

1.1 Discrete Dynamical Systems

TD networks are models of discrete dynamical systems (DDS) with discrete observations and actions. Any DDS is fully specified by three things: $\mathcal{O}$, $\mathcal{A}$, and $\mathcal{P}$. At

each discrete time step, a DDS emits an observation o_t from a discrete set of output symbols $\mathcal{O}$ and accepts an action a_t from a set of discrete input symbols $\mathcal{A}$. Based on all preceding events and a_t, the DDS probabilistically emits a new observation o_{t+1}. The sequence of actions and observations that begins at time 0 (o_0) and ends at time t (a_t) is called the history of the system at time t, or h_t. The symbol o_{t+1} is chosen probabilistically from a distribution conditioned on h_t. $\mathcal{P}$ is the probabilistic mapping from a history to an observation; $\mathcal{P}_h^o$ corresponds to $Pr(o_{t+1} = o|h_t = h)$.

Any representation of a DDS requires some structure in $\mathcal{P}$, some way to summarize this function that can generate infinite sequences of experience. Chapter 2 contains discussion of some common representations and the constraints and assumptions they imply on $\mathcal{P}$.

A *sufficient statistic* or *state* is any lossless summary of the history h_t. In the literature, the term "state" is sometimes used quite loosely. For example, Russell and Norvig imply that future experience can depend on the previous state, the previous two states, or the previous n states [Russell and Norvig, 2003]. Their definition of state does not mean sufficient statistic of the system.

1.2 Predictive Representations

Predictive representations are a novel paradigm used to create models of DDSs [Rivest and Schapire, 1990; Littman et al., 2002; Jaeger, 1998]. In *predictive representations*, the model is a collection of predictions about future experience and the mechanisms for updating those predictions.

If different histories have identical probability distributions over all future experience, they correspond to the same state. The implied assumption of predictive

representations is that equivalence of probabilities over all future experience can be summarized using predictions about a carefully selected subset of future experience. These predictions would then be a sufficient statistic.

1.3 Temporal-Difference Networks

Temporal-difference (TD) networks are a formalism for expressing and learning grounded knowledge about dynamical systems. A TD network is a predictive model that combines aspects of temporal-difference methods, predictive representations, and neural networks.

Knowledge is *grounded* if it can be directly related to experience: it is in terms of actions the agent can choose and symbols the agent can observe. Knowledge is not grounded when represented in some abstract sense that is understood by the designer and only indirectly related to the agent.

TD networks are an extension to existing work on temporal-difference methods. The idea of conventional TD methods is to "learn a guess from a guess", where both guesses are predictions about the same event made at different times. TD networks generalize this idea and learn from predictions of different events made at different times.

A TD network is a generative model of a discrete dynamical system. In a TD network, each prediction is the answer to some question about future interaction with the DDS. These questions may take a variety of forms, although most generally they are some function of future predictions and observations.

1.4 TD Network Overview

A *TD network* is a network of nodes, each representing a single scalar prediction. The nodes are interconnected by links representing the TD relationships among the predictions and to the observations and actions. These links determine the extensive semantics of each prediction—its desired or target relationship to the data. They represent what the agent should predict about the data as opposed to how the agent should predict it. These links determine a set of *questions* being asked about the data, and accordingly are called the *question network*.

Independent of the question network, a separate set of interconnections determine the actual computational process—the calculation of the predictions for each node at each time step. This process provides the *answers* to the questions, and accordingly is called the *answer network*. The question network provides targets for a learning process that shapes the answer network.

1.4.1 Question Networks

Figure 1.1 shows three suggestive examples of potential question networks. Assume the systems being modeled by these networks have two observations, $\mathcal{O} = \{0, 1\}$. The boxes at the top of each question network represent the observation bit o_{t+1}.

The node labeled 1 is directly connected to o_{t+1} and represents a question or prediction of the probability that the observation bit will be 1 on the next time step. For now, let the prediction of node i at time t be y_t^i and the TD target for y_t^i is z_t^i. The target for Node 1 can be described as $z_t^1 = Pr(o_{t+1} = 1)$.

The node labeled 2 is a prediction of the expected value of Node 1 on the next step, $z_t^2 = E(y_{t+1}^1)$. The extensive definition of Node 2's prediction is the probability

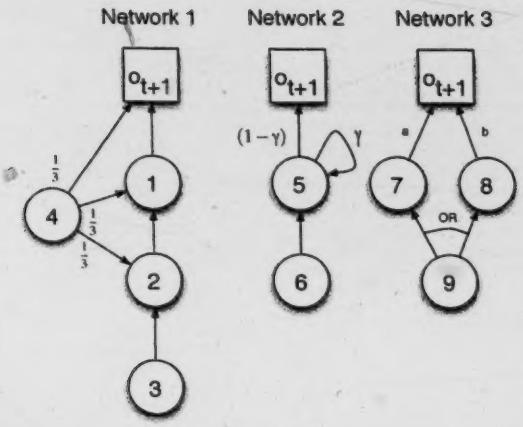

Figure 1.1: Three example question networks. The observation is represented as a box at the top while actual nodes of the TD network, corresponding each to a separate prediction, are below.

that the first observation bit will be 1 in two time step, $Pr(o_{t+2} = 1)$.

Node 3 similarly predicts the expected value of Node 2 at the next time step $z_t^3 = E(y_{t+1}^2)$. To *unroll* a question is to examine the implications of its questions to identify their extensive definition. If Node 3's question is unrolled, it can be interpreted as $E(y_{t+2}^1)$ or $Pr(o_{t+3} = 1)$.

Node 4 is more complicated, it can be thought of as the expected average value of the three nodes it is pointing to. The numbers on the links indicate the weight of each connection. The target for Node 4 is $z_t^4 = \frac{(E(y_{t+1}^2)+E(y_{t+1}^1)+Pr(o_{t+1}=1))}{3}$. Unrolled, the question is $\frac{(E(y_{t+2}^1)+Pr(o_{t+1}=1)+Pr(o_{t+2}=1))}{3}$ or $\frac{(Pr(o_{t+1}=1)+Pr(o_{t+2}=1)+Pr(o_{t+3}=1))}{3}$.

Node 5 is a conventional TD prediction, in this case of the future γ discounted sum of the observation bit. Its target is the familiar TD target, the data bit plus the node's own prediction on the next time step (with weightings $(1-\gamma)$ and γ respectively). The immediate target is $z_t^5 = (1-\gamma)o_{t+1} + \gamma y_{t+1}^5$. Unrolled, the

target is:

$$(1-\gamma)\sum_{i=0}^{\infty}\gamma^i o_{t+i+1}$$

Node 6 predicts the expected value of Node 5 at the next time step $z_t^6 = E(y_{t+1}^5)$, or unrolling the question further it is:

$$(1-\gamma)\sum_{i=0}^{\infty}\gamma^i o_{t+i+2}$$

Nodes 7 and 8 predict the probability of the observation bit being 1 *if* particular actions a or b are taken respectively. In target notation, $z_t^7 = Pr(o_{t+1} = 1|a_t = \mathcal{A}^a)$ and $z_t^8 = Pr(o_{t+1} = 1|a_t = \mathcal{A}^b)$. If the action at time t does not match the condition of the prediction, the prediction will not have a valid target.

Node 9 is a prediction of whether y_{t+1}^7 or y_{t+1}^8 will predict 1. The "or" can be defined arbitrarily, assume here it is the maximum of the two values. The TD target for Node 9 is $z_t^9 = max[E(y_{t+1}^7), E(y_{t+1}^8)]$. Unrolled, Node 9 is asking a question about the observation, $max[Pr(o_{t+2} = 1|a_{t+1} = \mathcal{A}^a), Pr(o_{t+2} = 1|a_{t+1} = \mathcal{A}^b)]$.

As the questions become more complex, they become harder to unroll and express in terms that can be understood by the designer. More nodes can be added, their extensive definitions are difficult to express but nevertheless are completely defined as long as the local TD relationships are clear.

1.4.2 TD Network Operation

The operation of a TD network can be summarized by five steps. A detailed explanation of the TD network operation is provided after these steps have been introduced.

1. Choose an action a_{t-1} and receive an observation from the environment o_t

2. Calculate the input vector $\mathbf{x}_t$ as a function of the previous predictions $\mathbf{y}_{t-1}$, the action just taken a_{t-1}, and the new observation o_t

3. Create the new predictions $\mathbf{y}_t = \boldsymbol{\sigma}(\mathbf{W}_t\mathbf{x}_t)$

4. Calculate the targets $\mathbf{z}_{t-1}$ for the previous predictions $\mathbf{y}_{t-1}$ using $\mathbf{y}_t$ and the observation o_t according to the question network's links and the action conditions

5. Update the weights W according to $(\mathbf{z}_{t-1} - \mathbf{y}_{t-1})$

The operation of the answer network is defined by a set of vector valued functions. Let $y_t^i \in [0,1]$, $i = 1, \ldots, n$ denote the prediction of the ith node at time step t. The column vector of predictions $\mathbf{y}_t = (y_t^1, \ldots, y_t^n)^T$ is calculated using a vector-valued function $\mathbf{u}$ with modifiable parameter $\mathbf{W}$:

$$\mathbf{y}_t = \mathbf{u}(\mathbf{y}_{t-1}, a_{t-1}, o_t, \mathbf{W}_t) \in \Re^n. \tag{1.1}$$

The function $\mathbf{u}$ corresponds to the answer network, with $\mathbf{W}$ being the weights on its links. In general $\mathbf{u}$ is an arbitrary function approximator, but for concreteness it is defined to be of a generalized linear form

$$\mathbf{y}_t = \boldsymbol{\sigma}(\mathbf{W}_t\mathbf{x}_t) \tag{1.2}$$

where $\mathbf{x}_t \in \Re^m$ is a feature vector, $\mathbf{W}_t$ is a $n \times m$ matrix, and σ is some n-vector activation function, in our experiments either the identity function or the S-shaped logistic function $\sigma(s) = \frac{1}{1+e^{-s}}$. The feature vector is an arbitrary function of the

preceding action, observation, and node values:

$$\mathbf{x}_t = \mathbf{x}(a_{t-1}, o_t, \mathbf{y}_{t-1}) \in \Re^m. \tag{1.3}$$

For example, $\mathbf{x}_t$ might have one component for each observation bit, one for each possible action (one of which is 1, the rest 0), and n more for the previous node values $\mathbf{y}_{t-1}$.

Each weight w^{ij} corresponds to the weight of input $\mathbf{x}_t^i$ for prediction $\mathbf{y}_t^j$. The update for each component w_t^{ij} of $\mathbf{W}_t$ is a fixed step size gradient descent update in least squares objective

$$w_{t+1}^{ij} - w_t^{ij} = \alpha(z_t^i - y_t^i)c_t^i \frac{\partial y_t^i}{\partial w_t^{ij}}, \tag{1.4}$$

where α is a step-size parameter, z_t^i is the TD target, and c_t^i is an update condition (described later).

The TD target z_t^i for y_t^i defined by the question network is a function of the successive predictions and observations. In vector form,

$$\mathbf{z}_t = \mathbf{z}(o_{t+1}, \tilde{\mathbf{y}}_{t+1}) \in \Re^n, \tag{1.5}$$

where $\tilde{\mathbf{y}}_{t+1}$ is like $\mathbf{y}_t$ in Equation 1.1, except calculated with the *old* weights before they are updated on the basis of $\mathbf{z}_t$:

$$\tilde{\mathbf{y}}_t = \mathbf{u}(\mathbf{y}_{t-1}, a_{t-1}, o_t, \mathbf{W}_{t-1}) \in \Re^n. \tag{1.6}$$

(This temporal subtlety also arises in conventional TD learning.) In addition to defining the TD targets, the question network also specifies action-conditions for predictions.

For example, Node 7 in Figure 1.1 predicts what the third observation bit will be *if* action a is taken. To arrange for such semantics there is a new vector $\mathbf{c}_t$ of *conditions*, c_t^i, indicating the extent to which y_t^i is held responsible for matching z_t^i, thus making the ith prediction conditional on c_t^i. Each c_t^i is determined as an arbitrary function c^i of a_t and y_t. In vector form,

$$\mathbf{c}_t = \mathbf{c}(a_t, \mathbf{y}_t) \in [0,1]^n. \tag{1.7}$$

For Node 7 in Figure 1.1, $c_t^7 = 1$ if $a_t = \mathcal{A}^a$, otherwise $c_t^7 = 0$.

The timing details may be clarified by writing the sequence of quantities in the order in which they are computed:

$$\mathbf{y}_t \, a_t \, \mathbf{c}_t \, o_{t+1} \, \mathbf{x}_{t+1} \, \tilde{\mathbf{y}}_{t+1} \, \mathbf{z}_t \, \mathbf{W}_{t+1} \, \mathbf{y}_{t+1}. \tag{1.8}$$

Prior work related to TD networks is presented in Chapter 2. We present experiments with TD networks in Chapter 3, followed by extensions to TD network specification in chapters 4 and 5. Finally, we conclude with an overall discussion of our results and future work in Chapter 6.

Chapter 2

Related Work

This chapter includes a survey of some prior work that is related to TD networks. Many representations can be used to model dynamical systems; a brief survey of the approaches that are most directly related to TD networks is presented. This survey includes methods that assume an underlying abstract state space, history-based methods, and existing predictive representations of state. This chapter includes a brief introduction to networks of interrelated predictors (artificial neural networks). TD networks are an extension of conventional temporal-difference methods of prediction, which are also briefly summarized in this chapter.

2.1 Temporal-Difference Methods of Prediction

Temporal-difference (TD) methods were formalized and studied by Sutton [1988] as a solution to the problem of making multi-step predictions of future events based on past experience. Previous to Sutton's formalization, well-understood techniques for learning predictions were trained using differences between predictions and the actual future outcomes. With TD methods, learning was applied using the differences

between temporally successive predictions. The example used by Sutton is that of a weatherman making a prediction on Monday about if it will rain on Saturday. The conventional approach would have been to wait until Saturday, observe if it rained, and then update the function to make better prediction on future similar Mondays. With Sutton's method, the weatherman would make a second prediction of Saturday's rain on Tuesday. The temporal-difference error between Monday and Tuesday's predictions could be used to improve predictions for similar Mondays. Sutton refers to the intuition of temporal-difference learning as "learning a guess from a guess".

TD methods are incremental, so they require fewer computational resources than their counterparts. In the weatherman example, after Tuesday has passed, no book-keeping or future information must be updated to improve the Monday prediction. With other approaches, extra work is required to keep track of all the predictions and then to finally update them when their target values are available. Sutton also claims that TD methods are more data efficient than the competing approaches; they converge faster and learn better predictions with limited data [Sutton, 1988].

2.2 Discrete Dynamical Systems

All discrete dynamical systems (DDSs) with discrete actions and discrete observations can be described by a tuple of three sets: $(\mathcal{O}, \mathcal{A}, \mathcal{P})$. Subscript notation is used to identify the time at which some event occurred; at each time step the agent chooses an action $a_t \in \mathcal{A}$ and the system returns an observation $o_{t+1} \in \mathcal{O}$. This sequence of actions and observations that begins at time 0 and ends at time t, $\{o_0 a_0 o_1 a_1, ..., o_t a_t\}$, is called the history of the system at time t, or h_t. The symbol

o_{t+1} is chosen probabilistically by the system from a distribution conditioned on its current history, h_t.

The observation function $\mathcal{P}$ is characterized by an *observation matrix* with an infinite number of rows and $|\mathcal{O}|$ columns. Each row in the observation matrix corresponds to some possible history, and each column corresponds to an observation from $\mathcal{O}$. The entries are probabilities: $\mathcal{P}_h^o = Pr(o_{t+1} = o | h_t = h)$.

For convenience, the rows of the matrix are ordered by length, the first row is the null history $\emptyset$, followed by all possible length one histories, then all possible length two histories, etc. The entry at row i, column j corresponds to the probability that observation o_j will be emitted by the system after observing history h_i.

Any representation of a DDS requires some structure in $\mathcal{P}$, some way to summarize this function that can generate infinite sequences of experience. A model is perfect if it is equivalent to $\mathcal{P}$. Another way to think about these representations is that they are generators of the observation matrix. If a model generates the same observation matrix as $\mathcal{P}$ then it is a perfect model of the DDS.

Sections 2.3.1-2.3.11 discuss some common representations and the constraints or assumptions they put on $\mathcal{P}$.

2.3 Modeling Dynamical Systems

When considering representations of dynamical systems, there are many different axes that can be used for comparison. One approach might work well with a small amount of data and a large amount of computation, while a different approach may use a relatively small amount of computation but needs much more data. There are a large number of other properties such as the ability to specify prior knowledge,

History	P(o^0)	P(o^1)
$\emptyset$ (h^0)	0	1
$\emptyset o^0 a^0$ (h^1)	0	1
$\emptyset o^0 a^1$ (h^2)	1	0
$\emptyset o^1 a^0$ (h^3)	.5	.5
$\emptyset o^1 a^1$ (h^4)	.1	.9
$\emptyset o^0 a^0 o^0 a^0$ (h^5)	1	0
$\emptyset o^0 a^0 o^0 a^1$ (h^6)	0	1
$\emptyset o^0 a^0 o^1 a^0$ (h^7)	0	1
$\emptyset o^0 a^0 o^1 a^1$ (h^8)	0	1
$\emptyset o^0 a^1 o^0 a^0$ (h^9)	1	0
$\emptyset o^0 a^1 o^0 a^1$ (h^{10})	1	0
$\emptyset o^0 a^1 o^1 a^0$ (h^{11})	1	0
$\emptyset o^0 a^1 o^1 a^1$ (h^{12})	.5	.5
$\emptyset o^1 a^0 o^0 a^0$ (h^{13})	.75	.25
$\emptyset o^1 a^0 o^0 a^1$ (h^{14})	.45	.65
$\emptyset o^1 a^0 o^1 a^0$ (h^{15})	0	1
$\emptyset o^1 a^0 o^1 a^1$ (h^{16})	1	0
$\emptyset o^1 a^1 o^0 a^0$ (h^{17})	1	0
$\emptyset o^1 a^1 o^0 a^1$ (h^{18})	.5	.5
$\emptyset o^1 a^1 o^1 a^0$ (h^{19})	.4	.6
$\emptyset o^1 a^1 o^1 a^1$ (h^{20})	.8	.2
...	...	...

Table 2.1: Example of the observation matrix for a DDS with 2 observations and 2 actions. Superscript notation refers to a particular element from a set. For example, o^i corresponds to the i^{th} element from $\mathcal{O}$ (starting at 0), and a^j corresponds to the j^{th} element from $\mathcal{A}$.

exact vs. approximate models, deterministic vs. stochastic models, etc., that can strengthen or weaken the case for using a certain approach depending on the context. The varying requirements of each situation dictate that there will be no approach that is right for all tasks. TD networks are no exception to this rule: while they are well suited to some tasks, they may be poorly suited to others.

The following sections include a brief introduction to various representations used for modeling dynamical systems. The focus of this discussion is representational power: some description of what types of DDSs the representation can model accurately.

2.3.1 Markov Decision Processes

The Markov decision process (MDP) representation assumes a fixed number of situations in the system, each deterministically emitting a unique symbol [Sutton and Barto, 1998]. There is a one to one mapping between the symbols in $\mathcal{O}$ and the situations in the system. This representation implies that $\mathcal{P}^o$ is conditioned only on the previous observation and action. The observation o_t is therefore a sufficient statistic in an MDP. This constraint reduces the number of unique rows in the observation matrix to be equal to the number of observations multiplied by the number of actions ($|\mathcal{O}| \cdot |\mathcal{A}|$). The size of an MDP model is the same size as the number of unique rows in the observation matrix. A graphical illustration of an MDP and its corresponding observation matrix are shown in Figure 2.1 and Table 2.2 respectively.

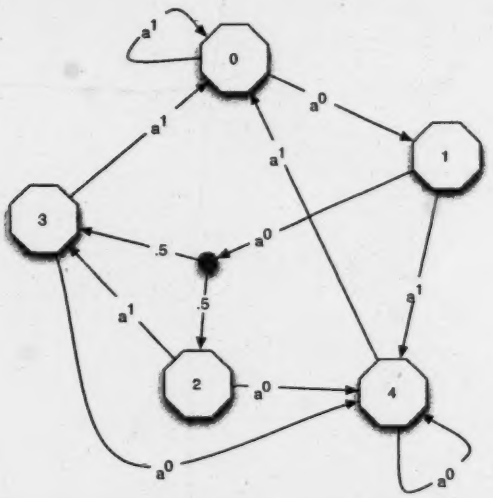

Figure 2.1: Graphical representation of a simple MDP with 5 states and 2 actions. The .5 links coming from state 1 correspond to a stochastic transition; taking action a^0 in state 1 can lead to either state 2 or state 3. The observation matrix for this MDP is shown in Table 2.2.

History	P(o^0)	P(o^1)	P(o^2)	P(o^3)	P(o^4)
$...o^0a^0$	0	1	0	0	0
$...o^0a^1$	1	0	0	0	0
$...o^1a^0$	1	0	.5	.5	0
$...o^1a^1$	0	0	0	0	1
$...o^2a^0$	0	0	0	0	1
$...o^2a^1$	0	0	0	1	0
$...o^3a^0$	0	0	0	0	1
$...o^3a^1$	1	0	0	0	0
$...o^4a^0$	0	0	0	0	1
$...o^4a^1$	1	0	0	0	0

Table 2.2: Example of the observation matrix for a DDS with 2 actions and 5 observations that can be modeled by an MDP. The "..." denotes that all preceeding histories are irrelevant. The corresponding graphical representation of this MDP is shown in Figure 2.1.

2.3.2 N^{th}-Order Markov Models

An n^{th}-order Markov representation (also known as a k-Markov or k-order Markov model) is a generalized version of the MDP representation. This representation requires that there are a fixed number of situations that can each be uniquely identified by the previous n observations and actions. An MDP is an n^{th}-order Markov model with $n = 1$. An n^{th}-order Markov model constrains the observation matrix such that there may be a unique row for each unique history suffix of length n. The number of unique rows in this matrix is $(|\mathcal{O}|^n \cdot |\mathcal{A}|^n)$. The size of this model is the same size as the number of unique rows in the observation matrix. The state in this representation is the length n suffix of the current history. A sample observation matrix for an n^{th}-order Markov model is shown in Table 2.3.

2.3.3 The System-Dynamics Matrix

The representations discussed in the following sections are distinct from those discussed previously because their models are more compact than their associated

History	P(o_0)	P(o_1)
$...o^0a^0o^0a^0$	.5	.5
$...o^0a^0o^0a^1$	1	0
$...o^0a^0o^0a^0$	.75	.25
$...o^0a^0o^0a^1$	0	1
$...o^0a^0o^1a^0$	0	1
$...o^0a^0o^1a^1$	1	0
$...o^0a^0o^1a^0$	.5	.5
$...o^0a^0o^1a^1$	0	1
$...o^0a^1o^0a^0$	1	0
$...o^0a^1o^0a^1$	1	0
$...o^0a^1o^0a^0$	0	1
$...o^0a^1o^0a^1$	1	0
$...o^0a^1o^1a^0$	.6	.4
$...o^0a^1o^1a^1$	.8	.2
$...o^0a^1o^1a^0$	0	1
$...o^0a^1o^1a^1$	1	0
$...o^1a^0o^0a^0$	0	1
$...o^1a^0o^0a^1$	.5	.5
etc.	...	...
$...o^1a^1o^1a^0$	0	1
$...o^1a^1o^1a^1$	1	1

Table 2.3: Example of the $\mathcal{P}$ function for a DDS with 2 actions and 2 observations that can be modeled by a 2^{nd}-order Markov model. The "..." denotes that all preceeding histories are irrelevant, it is only the suffix of the history that matters.

observation matrices. These models can generate an observation matrix with a near infinite number of unique rows, but cannot generate all possible observation matrices. To accurately discuss the constraints on $\mathcal{P}$ that are induced by these representations, it is necessary to further develop our understanding of the observation matrix and the complexity of dynamical systems.

Originally introduced by Singh *et al.* [2004], the *system-dynamics vector* is a theoretical construct that can be used to represent any discrete dynamical system. The observation matrix is an isomorphic transformation of the systems-dynamics vector. The systems-dynamics vector d has infinite length, each element corresponds to the probability of the DDS emitting a sequence of k observations given a particular

sequence of $k-1$ actions. All sequences of lengths $1, 2, 3, \ldots, n$ are included in this vector.

Each element in d has the form $\{o_0 a_0 o_1 a_1 ... a_{k-2} o_{k-1}\}$, corresponding to $Pr(o_0 o_1 \cdots o_{k-1} | a_0 a_1 \cdots a_{k-2})$. The probability of the full sequence is the product of several rows in the observation matrix:

$$Pr(o_0 o_1 \cdots o_{k-1} | a_0 a_1 \cdots a_{k-2}) = Pr(o_0|\emptyset) \times Pr(o_1|o_0 a_0) \times \cdots \times Pr(o_{k-1}|o_0 a_0 o_1 a_1 \cdots a_{k-2})$$

Any element (sequence) from d can be interpreted as a collection of histories, $e = \{\emptyset, o_0 a_0, o_0 a_0 o_1 a_1, \ldots, o_0 a_0 o_1 a_1 \cdots a_{k-2} o_{k-1}\}$. These histories are labeled $\{(\emptyset = e^0), (o_0 a_0 = e^1), \ldots, (o_0 a_0 o_1 a_1 \cdots a_{k-2} = e^{k-1})\}$. The element in d can be expressed as:

$$Pr(o_0 o_1 \cdots o_{k-1} | a_0 a_1 \cdots a_{k-2}) = \prod_{i=0}^{k-1} \mathcal{P}_{e^i}^{o_i}$$

The systems-dynamics vector d has an infinite number of entries that can be thought of as columns; the observation matrix has an infinite number of rows. These ideas can be combined to construct a new matrix with an infinite number of rows and columns, known as the *systems-dynamics matrix* $\mathcal{D}$ [Singh et al., 2004]. Like the observation matrix, each row of $\mathcal{D}$ corresponds to a particular history. Like d, each column corresponds to a sequence of observations and actions. In the literature, these sequences are called futures f or *tests*. Each entry $\mathcal{D}_j^i$ corresponds to the probability of observing the observations in test f^i given the history h^j and the action sequence of f^i. The first $|\mathcal{O}|$ columns of the matrix correspond to the one-step tests, the probability of seeing each observation given the history, exactly as in the observation matrix.

History	o^0	o^1	$o^0a^0o^0$	$o^0a^0o^1$	$o^0a^1o^0$	$o^0a^1o^1$	...
$\emptyset(h^0)$	x	x	x	x	x	x	x
h^1	x	x	x	x	x	x	x
h^2	x	x	x	x	x	x	x
h^3	x	x	x	x	x	x	x
h^4	x	x	x	x	x	x	x
h^5	x	x	x	x	x	x	x
h^6	x	x	x	x	x	x	x
...	...	...	...	...	...	...	...

Table 2.4: Sample systems-dynamics matrix for a DDS with 2 observations and 2 actions. In the columns of this matrix, the notation $o^ia^jo^k...$ is short form for the test $Pr(o_{t+1} = o^i, o_{t+2} = o^k | a_t = a^j)$, given the history corresponding to the row.

The system-dynamics matrix is directly computable from d or the observation matrix and therefore contains no more information than either of these representations. The advantage of considering $\mathcal{D}$ is that it has certain properties that allow for better analysis of the complexity of a DDS. Singh *et al.* define the *linear dimension* of a DDS as the *rank* of its system-dynamics matrix [Singh et al., 2004]. The linear dimension of $\mathcal{D}$ is a measure of the complexity of a DDS, so a model that can accurately represent a given system should have complexity that is a function of the linear dimension. For example, because the maximum number of unique rows in the observation matrix for an n^{th}-order Markov model is $k = (|\mathcal{O}|^n \cdot |\mathcal{A}|^n)$, $\mathcal{D}$ cannot have rank greater than k. Therefore, the linear dimension of any system that can be represented by an n^{th}-order Markov model cannot be greater than k.

2.3.4 Partially Observable Markov Decision Processes

Partially observable Markov decision processes (POMDPs) are like MDPs in the sense that they require n unique underlying situations in the system [Littman, 1996]. These situations are called *nominal states*. In the POMDP model, nominal states

do not emit symbols deterministically. Instead, each nominal state has a stochastic function to determine which symbol is emitted. A POMDP model maintains a probability distribution over how likely it is that the system is in each of the nominal states. This distribution is known as the *belief state*. POMDPs can represent systems that cannot be represented by any fixed-length history model. An example POMDP with deterministic transitions and observations is shown in Figure 2.2.

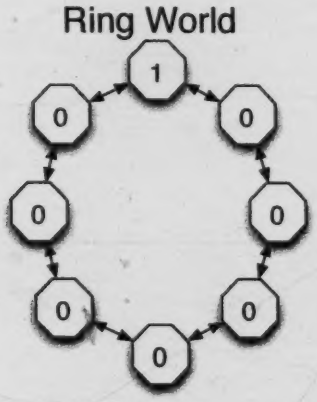

Figure 2.2: An indefinite-memory POMDP problem, the eight-state deterministic ring world. There are two actions in this world, *left* and *right*. *Right* advances in clockwise rotation while *left* advances in counter-clockwise rotation. Prediction methods using a finite length history will lose localization after some number of transitions back and forth between the states that emit observation 0.

Singh et al. have shown that the system-dynamics matrix $\mathcal{D}$ that can be generated by a POMDP with k nominal states can have rank no greater than k. Therefore, a POMDP with k nominal states cannot model a dynamical system with linear dimension greater than k [Singh et al., 2004]. The converse of this result is not true; there are DDSs with finite linear dimension that cannot be modeled by any POMDP [Jaeger, 1998].

2.3.5 Predictive Representations

The DDS representations above all require that the behaviour of the DDS can be described in terms of either some abstract space of situations or some suffix of history. Predictive representations instead characterize the state of the system in terms of questions about future sequences of actions and observations. Much of the work on predictive representations has been inspired by Rivest and Schapire's work [1994] on inferring the structure of deterministic finite automata. Rivest and Schapire were the first to propose that the state of a deterministic system could be represented by a vector of predictions about future tests that could be performed in the system.

There are several variations of predictive representations: most notably predictive state representations (PSRs) and observable operator models [Littman et al., 2002; Jaeger, 1998]. PSRs and observable operator models are similar in many ways, and are even equivalent under certain conditions [Singh et al., 2004]. For clarity, our focus will be specifically on Littman *et al.*'s PSR model.

2.3.6 Linear Predictive State Representations

The linear predictive state representation (linear PSR) is derived directly from the system-dynamics matrix $\mathcal{D}$. If $\mathcal{D}$ has finite rank k, there exist k linearly independent columns in $\mathcal{D}$ that can be used to generate the rest of $\mathcal{D}$. In a linear PSR, the tests that correspond to these k linearly independent columns from $\mathcal{D}$ are known as *core tests* [Littman et al., 2002]. The PSR model explicitly estimates the probability that each core test will succeed if its action sequence is chosen. Because a PSR is derived directly from $\mathcal{D}$, there is a PSR model with k core tests for every DDS with

finite rank k [Singh et al., 2004]. This result means that there is a PSR that can represent any n^{th}-order Markov model and POMDP, while the converse is not true.

2.3.7 TD Networks and Predictive State Representations

TD networks and PSRs have many similarities. Both methods are grounded, predictive representations. Both methods learn their model parameters from experience with the system. The work in both areas has progressed very quickly, and both representations have been used in various ways. Even so, there are some fundamental differences between TD networks and PSRs.

One clear difference is the type of tests that are used in each model. In the linear PSR literature, tests are for an entire sequence:

$$Pr(o_{t+1} = o^i, o_{t+2} = o^j, \cdots, o_{t+k} = o^x | a_t = \mathcal{A}^a, a_{t+1} = \mathcal{A}^b, \cdots, a_{t+k-1} = \mathcal{A}^f)$$

In TD networks, the questions are about the observation at the end of a sequence, similar to the e-tests in nonlinear PSRs [Rudary and Singh, 2004]:

$$Pr(o_{t+k} = o^k | a_t = a^a, a_{t+1} = \mathcal{A}^b, \cdots, a_{t+k-1} = \mathcal{A}^f)$$

The information that can be represented by these predictions is different. The implications of these differences are not completely clear.

To understand further differences between TD networks and PSRs, it is important to understand how these methods work.

Linear PSRs

This section presents a high level explanation of the linear PSR representation. For additional details, see the PSR literature [Littman et al., 2002; Singh et al., 2003; Singh et al., 2004; James and Singh, 2004; Wolfe et al., 2005].

Recall that a PSR representation assumes the existence of a maximal set of linearly independent columns in the system-dynamics matrix corresponding to core tests. If the core test values are known, the probability of all other tests can be calculated as a linear combination of the core tests.

The vector of core test probabilities is called Q. For every test f, there is weight vector w^f such that the probability of f succeeding can be calculated as a linear function of Q ($f_t = Q_t \cdot w^f$). Consider a single core test, q. There is a test f^{aoq} which is a one time step extension of q: the probability of the system generating the observation o following the action a and then the test q succeeding. There is also a test f^{ao}, the probability of seeing o after taking action a. Both f^{aoq} and f^{ao} are linearly calculable from Q ($f_t^{aoq} = Q_t \cdot w^{f^{aoq}}$ and $f_t^{ao} = Q_t \cdot w^{f^{ao}}$). The properties of the system-dynamics matrix (and Bayes' rule) allows the value of q at the next time step to be calculated:

$$q_{t+1} = \frac{f_t^{aoq}}{f_t^{ao}}$$

The core test probabilities can each be updated by dividing their appropriate one step extension test by the appropriate one step test. The parameters that a PSR learns are the weights $w^{f^{aoq}}$ for all core tests in Q and $w^{f^{ao}}$, the one step tests. The PSR representation requires that for every ao pair, there are two scalars for each $q_t \in Q_t$ that can be calculated linearly from Q_t. These scalars can then be

divided to calculate the value of q_{t+1}.

TD Networks

TD networks are not derived directly from the system-dynamics matrix and therefore the update procedure of the answer network is more difficult to analyze. The problem is compounded by the general nature of the TD network; there is no single "correct" update function, input representation, or question network.

Consider a simplified TD network update procedure where the input vector does not contain any direct information about the action and observation most recently taken. Instead, the input vector at time t is the prediction vector from the previous step ($\mathbf{x}_{t+1} = \mathbf{y}_t$). A bias term for the input representation can be used by adding a "null" question whose answer is always 1. Also, instead of using a single weight matrix W for calculating predictions, a different set of weights will be used for each $a_{t-1}o_t$ pair. Assuming that the σ activation function is the identity function, each prediction $y^i \in \mathbf{y}$ can be calculated as:

$$y^i_{t+1} = \mathbf{y}_t \cdot w^{ao}$$

At this level, the difference between the mechanics of a PSR and a TD network is clear. To calculate the value of a core test a PSR model computes both the values of the appropriate *ao* test and the one step extension to the core test, then divides these two numbers. In a TD network, the answers for the next time step are calculated directly as a linear function of the answers from the current time step. The derivation of the PSR update falls out directly from the definition of the core tests. The exact implications of the TD network update is less clear and is an active

area of investigation.

Discussion

TD networks and PSRs are two distinct representations and algorithms for learning predictive models. Although their predictive nature makes them similar, it is still unclear exactly what their relationship is. Is one of these two approaches superior? Are there certain problem classes in which each method has its strengths and weaknesses? In the future, will these two methods become more closely associated? We think that both methods will thrive and future work will further bridge the gap between them.

2.3.8 Nonlinear Predictive State Representations

Nonlinear PSRs do not require that the core tests be a maximally linearly independent set. Relaxing this constraint implies that one step extension to the core tests may not be a linear function of the core tests. The one step extension tests are instead calculated using some non-linear function [Rudary and Singh, 2004]. While the tests in a linear PSR predict the probability of a sequence of observations given a sequence of actions, other tests are possible. Nonlinear PSRS are based on *e-tests*, which predict the final observation given a sequence of actions [Rudary and Singh, 2004]. In their work, Rudary and Singh showed that in certain deterministic dynamical systems, nonlinear PSR models can be exponentially smaller than the equivalent linear PSR or POMDP model. It is currently unclear how nonlinear PSRs can be learned or applied in general.

2.3.9 Schemas

Schemas are a DDS representation, originally introduced by Drescher [1991]. Drescher's original work was recently extended by Holmes and Isbell [2005]. This representation was introduced as a way to model a DDS where the observations come as a vector $\vec{o_t}$, each element of the vector represents the discrete value of a sensor. In this representation, the model is a collection of action-conditional predictors called *schemas*. Each schema predicts the observation vector at time $t+1$. To improve their discriminative power, each schema has a filter or rule called a *context* that is only active if certain conditions are true of $\vec{o_t}$. As the agent gets more experience with the DDS, new schemas are added and the context of existing schemas are refined to make them more reliable. This agent also adds virtual sensors called *synthetic items* which represent hidden state. If some sensor cannot be predicted reliably, the agent assumes that there is some hidden variable (the synthetic item) that, if known, could help with prediction. Over time, the schemas will use and predict synthetic items just as they do regular observations.

Schemas are intended for deterministic DDSs – environments where $\mathcal{P}$ is such that, given some history h_t, the DDS will always generate the same sequence of observations for a particular action sequence. It is not clear how successfully the schema approach can model stochastic systems. There is no theoretical analysis of the potential representational power of the schema system.

2.3.10 Variable Length Memory Methods

Several representations have been proposed to model DDSs using variable length memories. These models are similar to the n^{th}-order Markov models discussed

above, but are distinct because not all histories are the same length. Variable length memory models are more efficient and tractable than n^{th}-order Markov models because long memories are used only in situations where shorter memories are not a sufficient statistic. Like n^{th}-order Markov models, variable length methods still cannot represent indefinite-memory problems like the ring world in Figure 2.2.

2.3.11 Other Methods

The preceding sections are a brief summary of the most relevant existing representations of dynamical systems. There are numerous other methods, each with their own assumptions and constraints. There are methods that only consider uncontrolled systems [Shalizi and Shalizi, 2004], methods for deterministic systems [Shen, 1993], methods that explicitly use a reinforcement signal in building a state representation [Ring, 1994; McCallum, 1995].

2.4 Artificial Neural Networks

Artificial neural networks (ANN) are an approach to information processing that was inspired by biological learning systems like the brain. Discussion about neurons and their behavior dates back as far as the 1940s and continues to be an active area of research [Mehrotra et al., 1997]. The ANN literature is sufficiently diverse and extensive that an attempt to summarize it here would be distracting to the reader. Instead, we present a general sketch of the neural network approach. For more information on neural networks, see Mehrotra *et al.*'s textbook [1997].

A neural network is a collections of nodes and weighted edges that transforms a vector of n inputs into a vector of m outputs. These networks are often structured

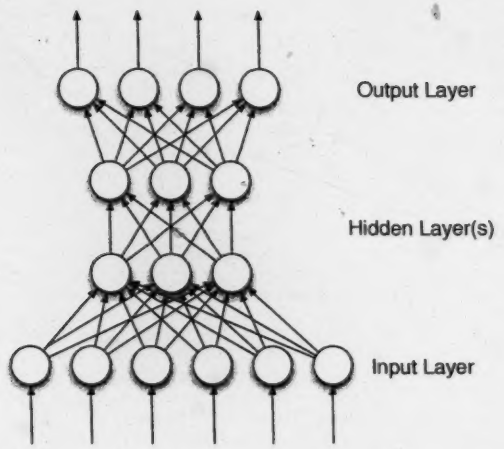

Figure 2.3: Artificial neural network with 6 input units, two hidden layers of 3 units each, and 4 output units.

using layers: there is an input layer, an output layer, and potentially several hidden layers in between. Information travels (usually directionally) along the edges between connected nodes. The information on the incoming edges to each node is summarized by an activation function and then transmitted through the node's outgoing edges. Figure 2.3 is a graphical representation of a neural network with two hidden layers. This type of network is known as a feed-forward neural network: the activations flow in a single direction.

Artificial neural networks that have backward connections are known as recurrent neural networks.

2.4.1 TD Networks and Artificial Neural Networks

It is natural to relate TD networks to artificial neural networks. Both are networks of interrelated predictors. ANNs are very general – they have been used in several configurations to solve many problems. The simplest way to relate TD networks to ANNs is to describe a TD network as an ANN with recurrent connections.

TD Networks as Recurrent ANNs

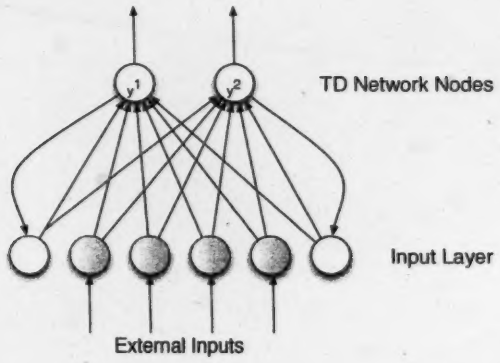

Figure 2.4: TD network with two predictive nodes drawn as a recurrent neural network. This figure describes the answer network and has no information about the question network.

A TD network can be thought of as a neural network with two layers, an input layer and an output layer. A sample TD network with two predictive nodes is shown in Figure 2.6. Each TD network node $y^i \in \mathbf{y}$ corresponds to a node in the output layer, and each element $x^j \in \mathbf{x}$ corresponds to a node in the input layer. The activation function can be whatever is desired. This is all that is required to understand the answer network of a TD network in terms of a recurrent ANN.

ANNs are often trained using input-output pairs: each output unit has a specific, well-defined target for each input vector. The target function in a TD network may include experience from future time steps and therefore does not fit this mold.

Other Recurrent Neural Networks

TD networks are not the first connectionist approach to make use of recurrent connections. There is a large body of literature that involves using recurrent ANNs for a variety of classification and prediction tasks.

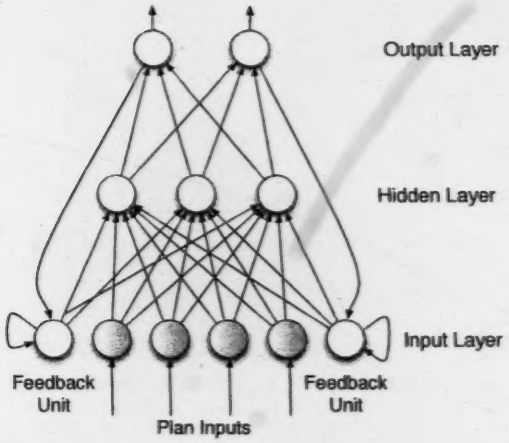

Figure 2.5: Example of Jordan's recurrent ANN with 4 external plan input units, 2 feedback units, 1 hidden layer of 3 units, and 2 output units.

Jordan [1986] proposed a multilayer feed-forward style network that connects output units back to certain special "feedback" units that operate in parallel to the external inputs units (called plan units). Jordan's networks can be used to produce a fixed sequence of outputs given a static input pattern.

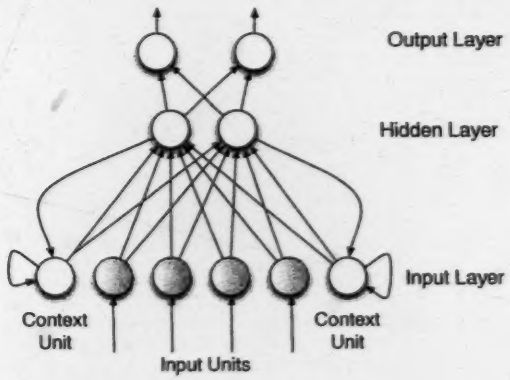

Figure 2.6: Example of Elman's recurrent ANN with 4 external plan input units, 2 feedback units, 1 hidden layer of three units, and 2 output units.

Elman [1988] suggested a similar network using recurrent connections from the hidden units rather than the output units. Elman's networks learn structure in time and were able to solve temporal XOR, learn about the structure in a sequence of

letters, and learn interesting knowledge in several other domains.

A variety of other recurrent network architectures also been proposed, including some fully recurrent variations [Rumelhart et al., 1986; Pineda, 1988; Williams and Zipser, 1989].

All of these models are trained using some variation of the error-propagation (back-propagation) learning algorithm [Mehrotra et al., 1997]. The back-propagation learning algorithm uses general purpose hidden units to correct to facilitate accurate predictions in the output units. This is different from the TD network learning algorithm where every unit has a well defined target function.

Discussion

Several types of recurrent artificial neural networks have been proposed in the past. To our knowledge, none of these approaches were intended to learn a predictive model of a dynamical system. None of these approaches learn TD style predictions. Although at first glance TD networks may seem similar to existing recurrent neural network architectures, comparison shows that TD networks are a novel, unique idea that cannot be directly equated with prior work in the recurrent connectionist literature.

Chapter 3

TD Network Experiments

In this chapter, we show that TD networks can be used to learn accurate, probabilistic predictions in a simple random-walk world. [1] When actions are introduced and the inter-prediction relationships aremade contingent on them, the usual learning-efficiency advantage of TD methods over Monte Carlo (supervised learning) methods becomes particularly pronounced. Finally, we demonstrate that TD networks can learn predictive state representations that enable exact solution of a non-Markov problem.

3.1 Error Reporting

The error of any particular prediction at one instant is not representative of the quality of the overall model; it is more meaningful to report error of predictions over the entire environment. Better yet, error could be reported over the entire environment weighted by the frequency that each state is visited. Finally, because

[1] This work was originally presented at the Advances in Neural Information Processing Systems Conference [Sutton and Tanner, 2005].

the TD network state representation is influenced by recent experience, the error should be averaged over these histories. The error of a node at some time step is defined as the root mean-squared error (RMSE) of that node's predictions over a window of some number of time steps (1000, for example). The errors observed within this window are a large enough sample to be representative of approximately regular experience in the environment. Let z_t^{*i} be the extensive, correct target for Node i.

Formally, the RMSE of Node 1 over a 1000 step window is defined as:

$$RMSE^1 = \sqrt{\frac{1}{|Window|} \sum_{t \in Window} (z_t^{*1} - y_t^1)^2}$$

The average error of an TD network with n nodes over a 1000 step window is then:

$$RMSE = \frac{1}{n} \sum_{i=1}^{n} RMSE^i$$

The targets (z_t^{*i}) are provided by asking an oracle what the unrolled answer to each question would be if the node's action sequence were performed from the current time. This oracle is not used for anything other than evaluation.

3.2 Random Walk

The experiments in this section use the 7-state random walk environment shown in Figure 3.1. Several variations of this world are used, including:

Fully observable stochastic walk This variation of the random walk environment has a single action. On each time step, the environment randomly moves

the agent either left or right with equal probability. This version of the random walk has been artificially made Markov, as shown in Figure 3.2. The observation matrix corresponding to $\mathcal{P}$ for this environment has exactly one row for each observation. Although the artificial state label is visible to the agent, the objective is to predict the observable bit.

Fully observable deterministic walk This environment is the observable stochastic walk with two actions. The first action deterministically moves the agent to the left, the other deterministically to the right. Again, the environment has been artificially labeled as in Figure 3.2.

7-state partially observable walk In this case their are two actions and the environment has not been artificially made Markov. Only the observation bit (1 or 0) is available to the agent.

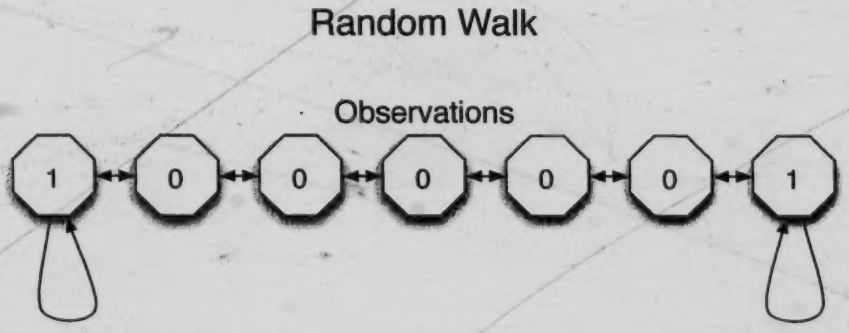

Figure 3.1: 7-state stochastic random walk world. The state transitions with equal probability to the left or the right on each time step.

3.2.1 Experiment 1: n-step Unconditional Prediction

In this experiment, the TD networks learned to predict the observation bit precisely n steps in advance, for $n = 1$, 2, 5, 10, and 25 in the observable stochastic walk

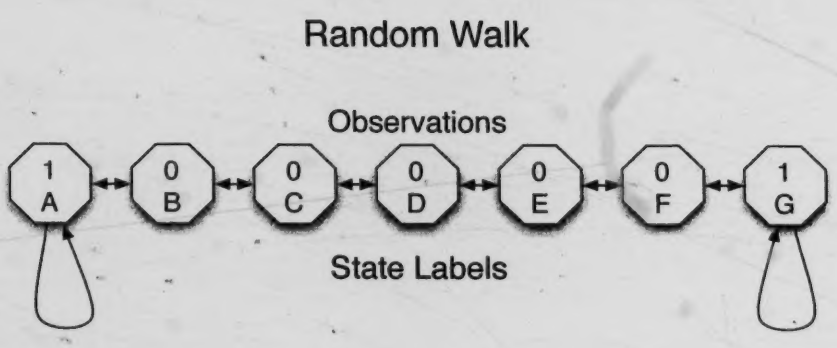

Figure 3.2: Fully observable 7-state stochastic random walk world used in our Markov experiments. The agent is able to observe the label of the state (bottom) and is trying to predict the observation in each step (top).

environment. In order to predict n steps in advance we also have to predict $n-1$ steps in advance, $n-2$ steps in advance, etc., all the way down to predicting one step ahead. This is specified by a TD network consisting of a single chain of predictions like the left column of Figure 3.3, but of length 25 rather than 5.

Random-walk sequences were constructed by starting at the center state and then taking random actions for 50, 100, 150, and 200 steps (100 sequences each).

A TD network and a corresponding Monte Carlo approach were both provided this data. The Monte Carlo method learned the same predictions, but learned them by comparing them to the actual outcomes in the sequence (instead of z_t^i in (1.4)). Both algorithms used feature vectors of 7 binary components, one for each of the seven states, all of which were zero except for that corresponding to the current state. Both algorithms formed their predictions linearly ($\sigma(\cdot)$ was the identity) and unconditionally ($c_t^i = 1\ \forall i, t$).

In an initial set of experiments, both algorithms were applied online with a variety of values for their step-size parameter α. Under these conditions neither algorithm was clearly better in terms of the root mean square error in their pre-

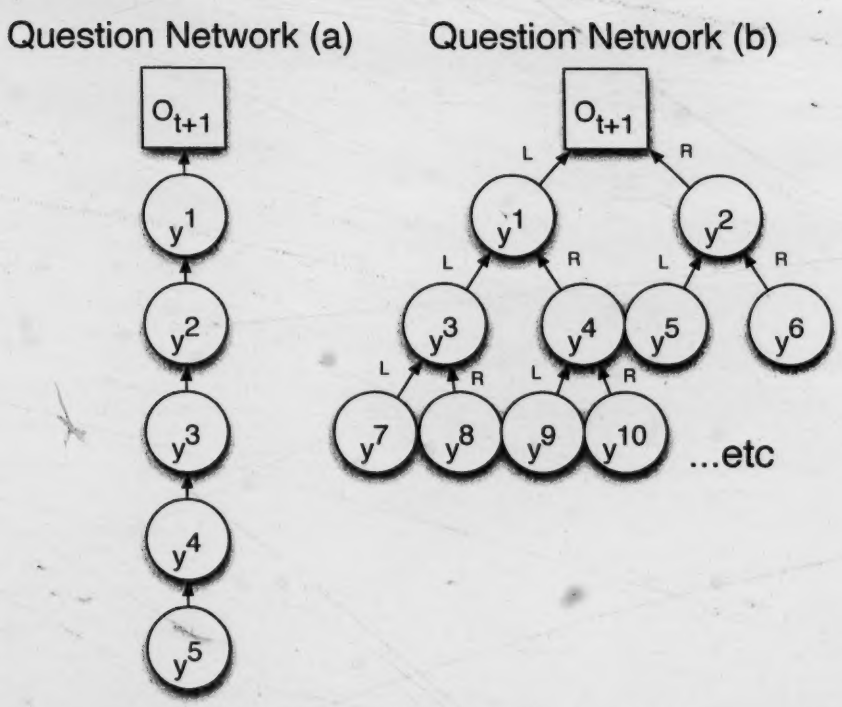

Figure 3.3: Question networks similar to those in our experiments. Question network (a) is a 5 step unconditioned "chain" question network, network (b) is a 3-level "tree" symmetric action-conditional question network.

dictions over the data sets. The difference is obvious when both algorithms were trained using batch updating, in which weight changes are collected "on the side" over an experience sequence and then made all at once at the end, and the whole process is repeated until convergence. Under batch updating, convergence is to the same predictions regardless of initial conditions or α value (as long as α is sufficiently small), which greatly simplifies comparison of algorithms. The predictions learned under batch updating are also the same as would be computed by least squares algorithms such as LSTD(λ) [Bradke and Barto, 1996; Boyan, 2002; Lagoudakis and Parr, 2003].

For 1-step predictions, the Monte-Carlo and TD methods performed identically,

Time	1-step	2-step		5-step		10-step		25-step	
Steps	MC/TD	MC	TD	MC	TD	MC	TD	MC	TD
50	0.205	0.219	0.172	0.234	0.159	0.249	0.139	0.297	0.129
100	0.124	0.133	0.100	0.160	0.098	0.168	0.079	0.187	0.068
150	0.089	0.103	0.073	0.121	0.076	0.130	0.063	0.153	0.054
200	0.076	0.084	0.060	0.109	0.065	0.112	0.056	0.118	0.049

Table 3.1: RMSE of Monte-Carlo and TD-network predictions of various lengths and for increasing amounts of training data over all states on the random-walk example. These results are for offline, batching training.

but for longer predictions a significant difference was observed. The RMSE of the Monte Carlo method increased with prediction length whereas for the TD network it decreased. The largest standard error in any of the numbers shown in the table is 0.008, so almost all of the differences are statistically significant. TD methods appear to have a significant data-efficiency advantage over non-TD methods in this prediction-by-n context (and this task) just as they do in conventional multi-step prediction [Sutton, 1988].

3.2.2 Experiment 2: Action-conditional Prediction

The advantage of TD methods should be greater for predictions that apply only when the experience sequence unfolds in a particular way, such as when a particular sequence of actions is chosen. In a second experiment the TD networks learned n-step-ahead predictions conditional on action selections in the observable deterministic walk environment. The question network for learning all 2-step-ahead predictions is shown in Figure 3.3b. The upper two nodes predict the observation bit conditional on taking a left action (L) or a right action (R). The lower four nodes correspond to the two-step predictions, e.g., the second lower node is the prediction of what the observation bit will be if an R action is taken followed by an L action.

These predictions are similar to the *e-tests* used in some of the work on predictive state representations [Rudary and Singh, 2004].

This experiment used a question network like that in Figure 3.3b except of depth four, consisting of 30 (2+4+8+16) nodes. The conditions for each node were set to 0 or 1 depending on whether the action taken on the step matched that indicated in the figure. The feature vectors were the same as the previous experiment. Now that predictions are conditioned on actions, the problem is deterministic and α can be set uniformly to 1. A Monte Carlo prediction can be learned only when its corresponding action sequence occurs in its entirety, but then it is complete and accurate in one step. The TD network, on the other hand, can learn from incomplete sequences but must propagate them back one level at a time. First the one-step predictions must be learned, then the two-step predictions from them, and so on. The results for online and batch training are shown in Tables 3.2 and 3.3.

	1-Step	2-Step		3-Step		4-Step	
Time Step	MC/TD	MC	TD	MC	TD	MC	TD
100	0.153	0.222	0.182	0.253	0.195	0.285	0.185
200	0.019	0.092	0.044	0.142	0.054	0.196	0.062
300	0.000	0.040	0.000	0.089	0.013	0.139	0.017
400	0.000	0.019	0.000	0.055	0.000	0.093	0.000
500	0.000	0.019	0.000	0.038	0.000	0.062	0.000

Table 3.2: *Online performance.* RMSE of the action-conditional predictions of various lengths for Monte-Carlo and TD-network methods on the random-walk problem.

As anticipated, the TD network learns much faster than Monte Carlo with both online and batch updating. Because the TD network learns its n step predictions based on its $n-1$ step predictions, it has a clear advantage for this task. Once the TD Network has seen each action in each state, it can quickly learn any prediction 2, 10, or 1000 steps in the future. Monte Carlo, on the other hand, must sample

Time Steps	MC	TD
50	53.48%	17.21%
100	30.81%	4.50%
150	19.26%	1.57%
200	11.69%	0.14%

Table 3.3: *Batch performance.* Average proportion of incorrect action-conditional predictions for batch-updating versions of Monte-Carlo and TD-network methods, for various amounts of data, on the random-walk task. All differences are statistically significant.

actual sequences, so each exact action sequence must be observed.

3.2.3 Experiment 3: Learning a Predictive State Representation

Experiments 1 and 2 showed advantages for TD learning methods in Markov problems. The feature vectors in both experiments provided complete information about the nominal state of the random walk. In Experiment 3, on the other hand, TD networks learned about a non-Markov version of the random-walk example, the partially observable random walk. In this case it is not possible to make accurate predictions based solely on the current action and observation.

As in the previous experiment, the TD network learned n-step predictions using action-conditional question networks of depths 2, 3, and 4. The feature vector $\mathbf{x}_t$ consisted of three parts: a constant 1, four binary features to represent the *pair* of action a_{t-1} and observation bit o_t, and n more features corresponding to the components of $\mathbf{y}_{t-1}$. The features vectors were thus of length $m = 11, 19, \text{and } 35$ for the three depths. In this experiment, $\sigma(\cdot)$ was the S-shaped logistic function. The initial weights $\mathbf{W}_0$ and predictions $\mathbf{y}_0$ were both 0.

Fifty random-walk sequences were constructed, each of 250,000 time steps, and presented to TD networks of the three depths, with a range of step-size parameters

α. The data performance measure was the RMSE of all predictions made by the networks (computed from knowledge of the task) and also the "empirical RMSE," the error in the one-step prediction for the action actually taken on each step. In all cases the errors approached zero over time, showing that the problem was completely solved. Figure 3.4 shows some representative learning curves for the depth-2 and depth-4 TD networks.

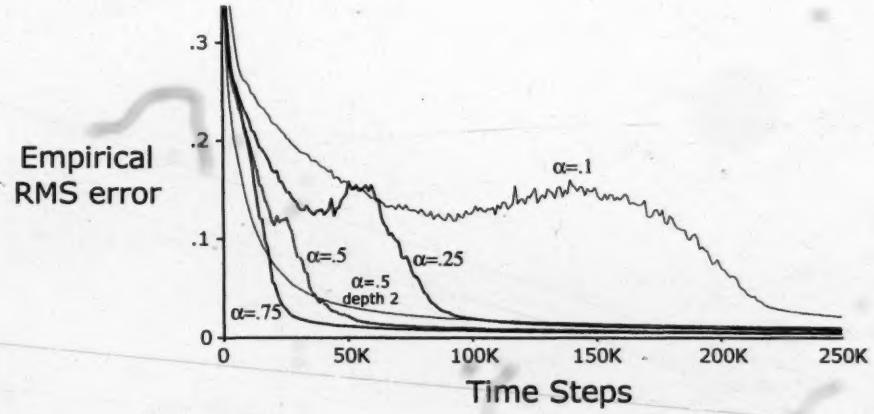

Figure 3.4: Prediction performance on the non-Markov random walk with depth-4 TD networks (and one depth-2 network) with various step-size parameters, averaged over 50 runs and 1000 time-step bins. The "bump" most clearly seen with small step sizes is reliably present and may be due to predictions of different lengths being learned at different times.

In ongoing experiments on other non-Markov problems, TD networks do not always find such complete solutions. Other problems seem to require more than one step of history information (the one-step-preceding action and observation), though much less than would be required using history information alone. Previous algorithms have also been found to be effective on some tasks but not on others [Singh et al., 2003; Rudary and Singh, 2004; James and Singh, 2004]. Our results

as a whole suggest that TD networks may provide an effective alternative learning algorithm for predictive state representations [Littman et al., 2002].

3.3 Conclusion

Our initial experiments with TD networks suggest a large set of possibilities for learning to predict, and in this chapter we have begun exploring the first few. Our results show that even in a fully observable setting there may be significant advantages to TD methods when learning TD-defined predictions. Our action-conditional results show that TD methods can learn dramatically faster than other methods. TD networks allow the expression of many new kinds of predictions whose extensive semantics is not immediately clear, but which are ultimately fully grounded in data.

Chapter 4

TD Networks with History

TD networks are not able to learn perfect models of some small, deterministic POMDPs. In this chapter, we extend TD networks by allowing the network-update process (the answer network) to depend on the recent history of previous actions and observations rather than only on the most recent action and observation. [1] This extension enables the solution of a larger class of problems than can be solved by the original TD networks or by history-based methods alone.

4.1 TD Network Counter-examples

TD networks are able to solve some, but not all of our small testing problems. Careful analysis has determined that the question networks that were used were sufficient to represent the appropriate model, so the issue must lie somewhere in the TD network specification.

Figure 4.1 presents a simple example of a task and question network for which

[1] The work in this chapter was originally presented at the International Joint Conference on Artificial Intelligence [Tanner and Sutton, 2005b].

the solution is representable but not learnable by TD networks without history. The cycle world consists of the four states shown on the left. The current state of the system cycles clockwise through the states. There is a single observation bit that is 1 at the top of the cycle, and 0 at all other times. On the right of Figure 4.1 is a question network which asks what the observation bit will be one, two, and three time steps in the future. Recall that at time t, $\mathbf{y}_t$ is calculated as a function of $(\mathbf{y}_{t-1}, a_{t-1}, o_t, \mathbf{W}_t)$. Assuming that the $\mathbf{y}_{t-1}$ is correct, there is a solution for the weights that will keep $\mathbf{y}_t$ correct at each successive time step. Unfortunately, $\mathbf{y}_{t-1}$ will never be correct; the solution exists but will not be found.

Recall that the question network specifies the source of target values for the answer network. At the start of training, $\mathbf{y}_{t-1}$ will likely be incorrect. There are no actions in this environment, so the current observation o_t is the only useful input feature in $\mathbf{x}_t$. For the network to become correct, it is necessary that some sequence of questions can eventually be answered, starting only with knowledge of o_t. Also note that when training begins, the only node with a valid target is y^1, because its target is not a prediction, but rather the grounded observable value of o_{t+1}. As

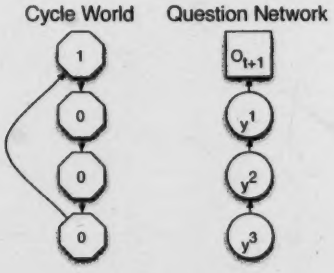

Figure 4.1: A counterexample for TD-network learning without history. On the left is a representation of the cycle world. This environment has four states that are cycled through deterministically. On the right is the associated question network. There are no actions in this world.

training progresses, the agent interacts with the environment and some answers will be learned using only the grounded observations. Eventually, the environment will reach a point where $o_t = 0$ and y_t^1 *should* be 1. The information that distinguishes this case from the case where $o_t = 0$ and y_t^1 *should* be 0 lies in a *correct* answer for y_{t-1}^2. Unfortunately, the target for y_{t-1}^2 is y_t^1. In this case, the cyclic dependency between the question network and the temporal flow of information eliminates the possibility of the TD network learning a correct solution.

Information flow dependency occurs when y_t^i critically depends on an input feature in $\mathbf{x}_t$ that corresponds to y_{t-1}^j, and the target z_{t-1}^j is a function of y_t^i. This dependency can be eliminated by providing additional input features to the TD network.

The cycle world is a problem in which there is a simple relationship between the observations and recent experience. Methods that try to directly learn such relationships are called history-based methods. We will consider history-based methods which predict o_{t+1} using a different variable for each unique k-length window of history where a k-length window of history is defined as $a_{t-k}o_{t-k+1}...a_{t-1}o_t$. In this case, a window of length 3 would be sufficient to uniquely identify each state of the system and thus would be able to make accurate predictions. Incorporating short history into the feature vector $\mathbf{x}_t$ of a TD network should allow the TD network to learn a correct solution to this problem. Figure 4.2 shows an example of a hybrid input vector that uses 3 time steps of history and 3 predictive nodes.

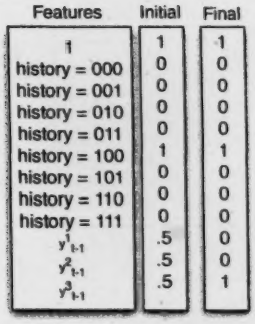

Figure 4.2: Input vector for cycle world with 3-step history and 3 levels of predictive nodes. On the left is the definition of each feature. The first feature is the bias term. The next 8 features correspond to the 8 distinct 3-step histories $\{o_{t-2}o_{t-1}o_t\}$ (not all are possible in this world). The final 3 features are the predictions from the previous time step. The middle vector is a sample input vector for the third state from the top of the cycle world at the start of learning. At this point, all of the predictions are at their initial value, .5. Finally, the rightmost vector is the input vector for the third state when learning is complete, all of the predictions are accurate.

4.1.1 Experimental Results

The hybrid approach is tested using a cycle world like that in Figure 4.1, except with six states instead of four. This size was chosen to clearly illustrate the effectiveness of different configurations of history and predictive nodes. Three different methods were tested in this domain: (1) TD networks as previously specified without history, (2) a simple history-based approach, (3) a combination of TD networks and history together. For each method, several values for the step size parameter were used; the best of these was used as the performance measure for that method. For each method and step size, the network was trained for at least one million time steps. The 1-step RMSE over the final 20 000 steps is used as an overall performance measure for each experiment.

The results in Figure 4.3 show that the simple history-based method only performed well when it had enough history to solve the problem exactly. TD networks

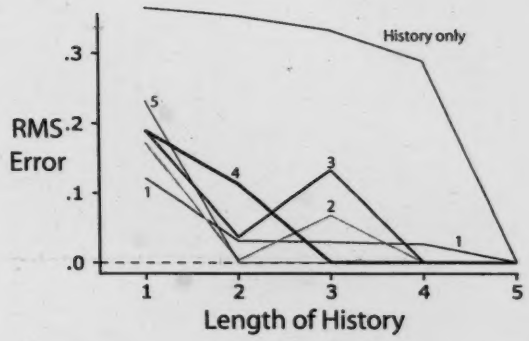

Figure 4.3: Performance on the 6-state cycle world of TD networks extend to incorporate various lengths of history. The different lines correspond to different depths of the question network, as indicated by the numeric label.

without history correspond to the data points with history length one. These TD network performances are better than history alone, but not as good as the TD networks augmented with history. It is also interesting to notice that the TD network is able to solve the problem with a much shorter window than the history-based method alone. This illustrates that our combined algorithm is not simply using history instead of the predictive representation, but rather is leveraging the history to learn a predictive representation. It is interesting that the performance of the various combinations of history and predictive nodes do not follow a clear pattern. For example, when there are 2 predictive nodes, it appears that 2 or 4 steps of history is better than having 3 steps. The minimum length of history required to exactly model the 6-state cycle world with 2 predictive nodes is a 4-step history. This means that the low error seen with 2 steps of history is a case of the TD network stumbling on a good approximate solution when it could not represent an exact solution (as discussed in Chapter 4.1.2).

4.1.2 Approximate Solutions

State	Observation	Sequence A	Sequence B
1	1	.83	.05
2	0	.00	.00
3	0	.02	.01
4	0	.03	.02
5	0	.06	.03
6	0	.15	.04
1	1	.83	.07

Table 4.1: Unstable approximate solutions learned by a single node TD Network on the 6-state cycle world. The predictions in sequence A are at a point in training where the TD network has found a very good approximate solution. The predictions in sequence B are from a different point in training. The behavior of the TD network oscillates indefinitely between producing solutions like sequence A and solutions like sequence B.

Approximate solutions to the cycle world can be learned by TD networks consisting of a single node. There is no way that a single predictive node can solve this problem perfectly, but it can achieve very low error in an unusual way. Table 4.1 shows two sequences of predictions that are made by a TD network at different stages of training. Neither sequence is stable, the TD network will oscillate between predictions like sequence A and predictions like sequence B.

The mean squared error of sequence A (over 1 trip around the cycle) is .0094 and sequence B is .151. The maximum likelihood model of this system based only on 1-step observations would predict 0 for the next step if 1 is observed, and would predict $\frac{1}{5}$ if 0 is observed. This model would have MSE=.133, better than sequence B but far worse than sequence A.

4.2 Indefinite-memory Problems

Introducing history to the TD network specification can eliminate cyclic dependencies and increase the class of problems where solutions can be learned with TD networks. There may be a tradeoff between predictive levels in the question network and lengths of history that are provided. From the cycle world example, it may not be clear that the hybrid approach is superior to a history-only approach.

There is a potentially large class of problems that cannot be represented with a history-only approach, but can be represented and solved by TD networks. Environments in this class are such that there is no finite length of history that can uniquely identify the current state of the environment. Problems in this class are called *indefinite-memory* problems.

One simple example of an indefinite-memory problem is the ring world shown in Figure 4.4. Because states B, C, D, and E are indistinguishable, there are sequences of actions that keep the environment in that subset of states and will eventually fill a fixed-length memory with useless information. In contrast, a TD network can model this environment, and can never be made to forget its location in the environment.

4.2.1 Experimental Results

We applied TD networks with various depths of question network and lengths of history to the 5-state ring world problem. The performance measure used was the same as in the previous experiments, except in this case averaged over 25 independent runs of 10 million time steps. The results are shown in Figure 4.5. As the history window increases, the history-only method more closely approximates the correct solution. This improvement seems to diminish as the history window gets

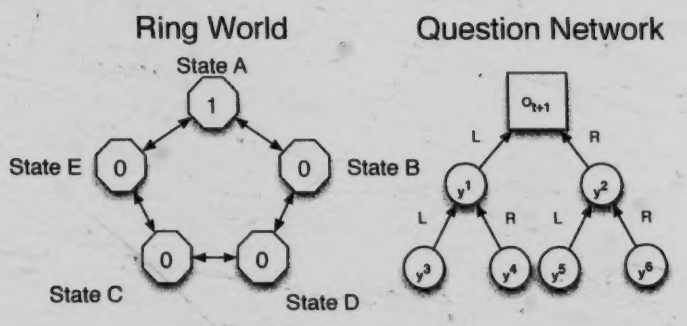

Figure 4.4: An indefinite-memory problem, the five-state deterministic ring world and an example question network of depth 2. There are two actions in this world, *left* or simply L and *right* R. *right* advances in clockwise rotation while *left* advances in counter-clockwise rotation. Prediction methods using a finite length history will lose localization after some number of transitions back and forth between the states that emit observation 0.

larger, and is further hampered by the fact that the number of unique histories grows exponentially with the length of the window. With the predictive approach, the problem is solved correctly with only 1 level of history and a predictive question network of depth 3.

Provided enough time, the TD network can learn a correct model of this environment without history, something which it could not do for the cycle world. This is puzzling given that these two problems seemed highly related, the cycle world seemingly even less complex than the ring world. In the ring world, actions have inverses which may eliminate the information flow dependencies that existed in the cycle world. In the ring world, the agent can incrementally learn more and more about the environment. In early training, the agent can orient itself when $o_t = 1$ because this observation uniquely identifies this state. As time passes, the agent can learn accurate 1-step predictions from that location. It can also learn 2-step predictions that involve leaving this position and then returning immediately. This process can continue until this chaining effect has allowed the agent to make accurate

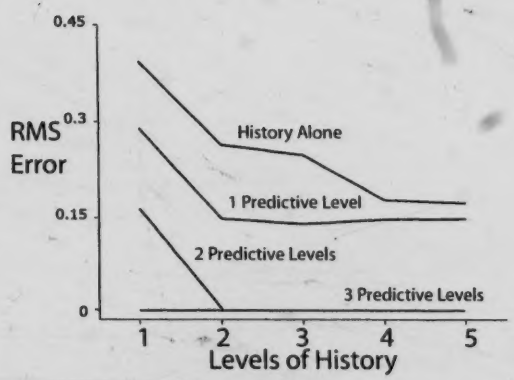

Figure 4.5: Performance on the 5-state ring world as a function of length of history and depth of question network. The history method suffers from diminishing returns as size of the history window increases. Learning also slows considerably because the number of unique histories that can be observed grows exponentially.

predictions from all positions in the ring.

4.3 Conclusions

We have presented a straightforward extension of TD networks to incorporate the strengths of history-based methods. The combination of history-based learning and TD network learning is more than putting two algorithms into one box and using the appropriate approach for a particular problem; the combined algorithm is stronger than either of its parts on their own.

Chapter 5

TD(λ) Networks

TD networks are similar to conventional TD(0) predictors, both algorithms use 1-step backups to train prediction units about future events. In conventional TD learning, the TD(λ) algorithm is often used to do more general multi-step backups of future predictions. In this chapter, we introduce a generalization of the 1-step TD network specification that is based on the TD(λ) learning algorithm, creating TD(λ) networks. [1] We present experimental results that show TD(λ) networks can learn solutions in more complex environments than TD networks. We also show that in problems that can be solved by TD networks, TD(λ) networks generally learn solutions much faster than their 1-step counterparts. Finally, we present an analysis of our algorithm that shows that the computational cost of TD(λ) networks is only slightly more than that of TD networks.

[1] The work in this chapter was originally presented at the International Conference on Machine Learning [Tanner and Sutton, 2005a].

5.1 Questions and Targets

In general, questions can be any function of future predictions or observations. In this section, consider the special case of question network in which each node i has a single target, either some other prediction or the observation at the next time step. This type of question network is called a *single-target* question network. This special case includes all of the question networks that were implemented in this research. The target of node i is the *parent* of i or $p(i)$. The later parents of node i: $\{p(p(i)), p(p(p(i))), ...\}$ are written in the short form $\{p^2(i), p^3(i), ...\}$.

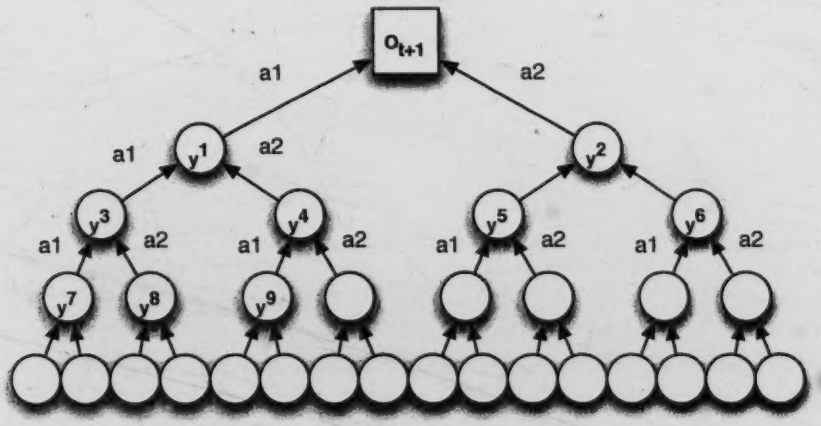

Figure 5.1: Symmetric action-conditional question network. The network forms a symmetric tree, with a branching factor equal to $|A|$. This example has depth $d = 4$. Some of the labels have been left out of this diagram for clarity, each of these nodes should have a label y^i and each is conditioned on some action.

In Figure 5.1 the parent of Node 9 is Node 4, $p(9) = 4$. The parent of Node 4 is Node 1 ($p(4) = 1$), so $p(p(9)) = p^2(9) = 1$. The third parent of Node 9, $p(p(p(9))) = p^3(9) = o$, the observation bit.

In Section 1.4.2, the target of a node was defined in a general sense. With single-target question networks, the target can be defined more specifically.

In these networks, the target for node i is:

$$z^i = y_{t+1}^{p(i)} \text{ or } o_{t+1} \tag{5.1}$$

Recall that each component w_t^{ij} of $\mathbf{W}_t$ is updated by the learning rule:

$$w_{t+1}^{ij} - w_t^{ij} = \alpha(z_t^i - y_t^i)c_t^i \frac{\partial y_t^i}{\partial w_t^{ij}} \tag{5.2}$$

where α is a step-size parameter and c_t^i corresponds to whether the action condition of the question was met.

Predictions are calculated using the logistical sigmoid function, so the exact weight update rule is:

$$\Delta w_t^{ij} = \alpha(z_t^i - y_t^i)y_t^i(1 - y_t^i)x_t^j c_t^i \tag{5.3}$$

5.2 TD(λ) Networks

The target function described in Equation 5.1 is correct for single-step TD(0) updates. Each prediction made at time t has a TD target that may become available at time $t+1$. This TD target may itself be a prediction of some other value that will be available at time $t+2$. Recall that predictions also have an extensive definition; it is possible to unroll the first prediction: to ask a question at time t about an event at time $t+2$. A question can be unrolled step by step until it is asking a question about the data, the observation bit. Each prediction made at time t is indirectly predicting several events at different moments in time and therefore has a different

target for each moment.

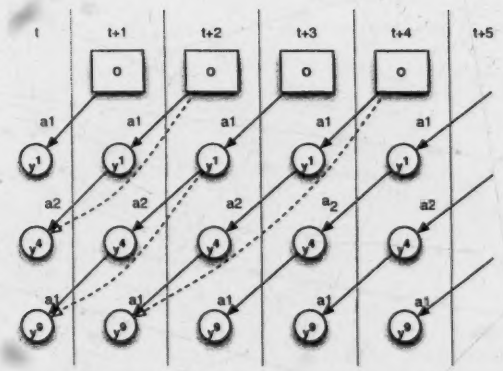

Figure 5.2: Extended target flow diagram for nodes {1,4,9} of question network in Figure 5.1. The links in this diagram show the flow of target values back toward the original predictions. The solid links are the 1-step TD targets for these predictions. The dashed links are a sample of the unrolled multi-step targets. The action-condition labeling have been omitted on the d dashed links to reduce clutter.

Using the parent function $p(i)$, the relationship between targets follows the structure of the question network. The first target for y_t^i comes directly from the 1-step TD relationship in the question network, and is simply z_t^i. The second target is recursively defined, it is the target of the parent of node i at the next time step, $z_{t+1}^{p(i)}$. Following this process, there is a k step sequence of targets for y_t^i: $z_t^i, z_{t+1}^{p(i)}, z_{t+2}^{p^2(i)}, \ldots, z_{t+k-1}^{p^{k-1}(i)}$, where $p^k(i)$ is the observation bit.

In order to keep notation a simple as possible, consider a single node i and a single starting time step t. Under these conditions, let the first target of a prediction be $z(0)$. The subsequent targets are $z(1), z(2), z(3), ..., z(k-1)$.

Consider the prediction Node 9 in Figures 5.1 and 5.2. In this case, $z(0) = y_{t+1}^4$, $z(1) = y_{t+2}^1$, and $z(2) = o_{t+3}$. Although each target in this sequence is a prediction of the same event o_{t+3}, each was calculated using different information. Our intuition is that the targets generated later in time may sometimes be more accurate than

the earlier targets. Some combination of the targets from this sequence may then be better than any particular single target in the sequence.

While the conditions of the predictions match the experience of the agent, this sequence of targets is available and valid. If the agent's experience diverges from the conditions of the question network, no further updates are performed. If multiple targets become available, any (or all) of these targets can be used for learning. As with TD(λ), we propose an exponentially weighted average of these targets. The multi-step weighted target for prediction y_t^i is denoted as v_t^i, where:

$$v_t^i = ((1-\lambda)\sum_{n=0}^{k-1}\lambda^n z(n)) + \lambda^k z(k-1) \tag{5.4}$$

The one-step target is given weight $(1-\lambda)$, the two-step target is given $(1-\lambda)\lambda$, the three-step target is given $(1-\lambda)\lambda^2$, etc. The last item in this sequence will receive all of the remaining weight (λ^k).

Ideally, we would like to use the blended target v_t^i in an update rule such as:

$$\Delta w_t^{ij} = \alpha(v_t^i - y_t^i)y_t^i(1-y_t^i)x^j \tag{5.5}$$

We desire an online, incremental algorithm where the value of v_t^i will not be available at time t. Some standard (and novel) tricks to allow this learning rule to be implemented incrementally using a variation of eligibility traces as with TD(λ). Pseudo-code for the TD(λ) networks learning algorithm is in Figure 5.3. The weight update rule in this algorithm achieves the behavior of Equation 5.5 using incremental updates of successive targets.

Because the predictions made within TD networks are of different events, im-

plementing eligibility traces is not as simple as with conventional TD(λ). Each prediction y_t^i needs its own eligibility trace. This makes our algorithm slightly more complicated than traditional TD(λ).

5.2.1 TD(λ) Network Learning Algorithm

```
Traces ← {}
for t = 0 to T do
  newTraces ← {}
  a ← chooseAction()
  o ← getObservation(a)
  x_t ← x(a, o, y_{t-1})
  y_t ← σ(W x_t)
  for (i, k) ∈ Traces do
    if checkCondition(p^{t-k-1}(i), a) == TRUE then
      if p^{t-k}(i) ≠ observation then
        z ← y_{t-1}[p^{t-k}(i)]
      else
        z ← o
      end if
      p ← y_{t-1}[p^{t-k-1}(i)]
      for w^j ∈ W[i] do
        w^j += α(z - p)p(1 - p)x_k^j λ^{t-k-1}
      end for
      if p^{t-k}(i) ≠ observation then
        newTraces ← newTraces ∪ (i, k)
      end if
    end if
  end for
  for i ∈ y do
    newTraces ← newTraces ∪ (i, t)
  end for
  Traces ← newTraces
end for
```

Figure 5.3: Pseudo-code for TD(λ) learning algorithm. The algorithm uses a boolean function $checkCondition(i, a)$ which will return true if the action a is consistent with the action condition of node i, and false otherwise.

5.2.2 TD(λ) Network Learning Algorithm Discussion

The TD(λ) learning algorithm keeps a record of predictions and whether the conditions of the unrolled definition of those predictions are consistent with later experience of the agent. At each time step, new targets become available for past predictions. By combining the temporal-difference $(\mathbf{y}_t - \mathbf{y}_{t-1})$ with historic information about the inputs to the answer network $(\mathbf{x}_{t-k})$, the weight vector $\mathbf{W}$ is updated (scaled by λ^{t-k-1}) towards the new target to improve the past prediction $\mathbf{y}_{t-k}$.

This algorithm has some interesting properties, controlled by the particular value of λ that is used.

If $\lambda = 0$, the first target of a prediction will get weight $0^0 = 1$, meaning that this first target will get the full weight of the update. For subsequent targets, $0^{t-k-1} = 0$ resulting in the update having no effect. This behavior is exactly the same as the previous 1-step TD network learning algorithm.

If $\lambda = 1$, each target available gets the full weight of the update, because $1^x = 1$. Each subsequent temporal-difference effectively overwrites the update made by the previous target. The net effect is that the last available target receives the full weight of the update and the intermediate targets receive no weight. If the prediction's conditions match exactly with the stream of experience, all of the weight will go to the grounded, unrolled target. If the stream of experience diverges from the conditions of the prediction, the weight of the update will go to some intermediate TD target. This behavior is analogous to a Monte Carlo style of update, with one important difference. In a Monte Carlo approach, an update would only occur if the conditions of the completely unrolled definition of a prediction were met. Nodes

at deeper depths would be exponentially less likely to receive updates, because the exact sequence of the conditions is less likely to occur. With TD(1), these predictions will always receive an update if their first condition is met.

Finally, if an intermediate value of λ is used, the weight of the updates are divided among the targets that become available. The remainder of the weight will always be assigned to the final available target.

5.3 Experimental Results

There are certain partially-observable environments for which a TD network solution exists, but the TD(0) learning algorithm cannot find it. The recursive nature of TD networks allow the occurrence of information flow dependencies between the question and answer networks. These information flow dependencies are a major obstacle when trying to learn a model of certain dynamical systems. One TD(0) solution to this problem is to augment the input vector $\mathbf{x}$ by including recent actions and observations in addition to the immediately previous action and observation. This recent history allows the TD(0) learning algorithm to solve problems that could not be solved without history. History also allows the TD(0) learning algorithm to solve existing problems faster than before.

Our hypothesis is that for some values of $\lambda > 0$, the TD(λ) network learning algorithm can solve this information flow dependency problem without adding additional information to the input vector.

TD(λ) networks are compared to TD(0) networks in three domains. It isn't clear exactly what is the best metric to compare one TD network learning algorithm to another; we will report the average RMSE of the answer network vs. amount of

data use to learn that model. This measure will illustrate both the speed of learning and the relative error of the models that are learned.

In each experiment, a variety of values were used for the step size parameters α, and the results presented are for whichever value of α performed best. In general, if any value of α could solve the problem, then all values of α that were used {.5, .25, .125, .0625} were able to solve the problem. Lowering α increased the amount of data required to learn a solution of the same quality. In each of these experiments, the initial weights in the answer network were set uniformly, $w_{ij} = \frac{1}{|\mathbf{x}|}$. Each environment (discussed further below) is started in the state where $o_t = 1$.

The first experimental results (Figure 5.4) are for the 6-state cycle world in Figure 4.1. The question network used for this experiment was a chain of 5 predictions like that in Figure 4.1. In Chapter 4.1, this problem could not be solved with TD(0) networks unless the input vector $\mathbf{x}$ is expanded to include recent history. TD(λ) networks can solve this problem without history.

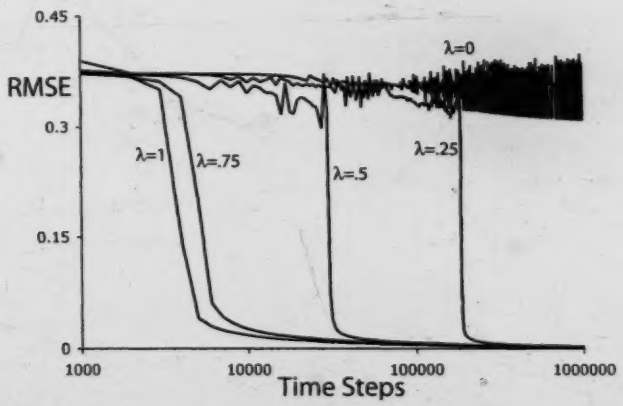

Figure 5.4: Learning curves of our learning algorithm for various values of λ on the 6-state cycle world. This chart represents the average RMSE over all of the nodes in the TD(λ) network as the amount of data is increased. Each data point in this graph is the average error of the network over 500 time steps. Note that the x-axis (amount of data) is an exponential scale. The cycle world is completely deterministic, so these results are for a single training run.

For all values of $\lambda > 0$, the TD(λ) network learning algorithm is able to find a solution. As λ increases, the amount of data required for learning decreases. A good model (RMSE $< .05$) is found with $\lambda = 1$ in under 5 000 steps. To learn an equivalent model, $\lambda = .75$ requires 7 000 steps, $\lambda = .5$ requires 32 000 steps, and $\lambda = .25$ requires 189 000 steps.

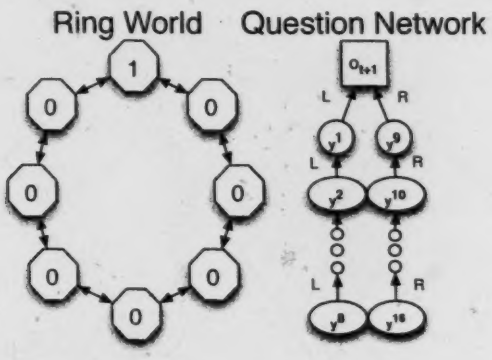

Figure 5.5: 8-state version of the ring world. On the left is a representation of the ring world. One of the states has an observation bit of 1, all of the others are 0. There are two actions in this world, one that moves the agent clockwise (call it 'right' or just R) and one that moves the agent counter-clockwise ('left' or L). The question network on the right side of this figure is a sparse action conditional network that can represent a solution to this world. This question network has 8 levels, at each level there is a question about action L and a question about action R.

The second experimental domain is the n-state ring world, shown in Figure 5.5. This domain is more complex than the cycle world because it has multiple actions. The actions used to generate experience for our experiments are chosen randomly. The results from testing our algorithm for various value of λ on the 5-node and 8-node versions of the ring world are shown in Figures 5.6 and 5.7 respectively.

It is important to experiment with the 5-state ring world to investigate the effect of λ on a problem that can be solved with TD($\lambda = 0$). For all of the $\lambda > 0$ values that were used, RMSE $< .05$ was achieved in under 10 000 time steps. As before,

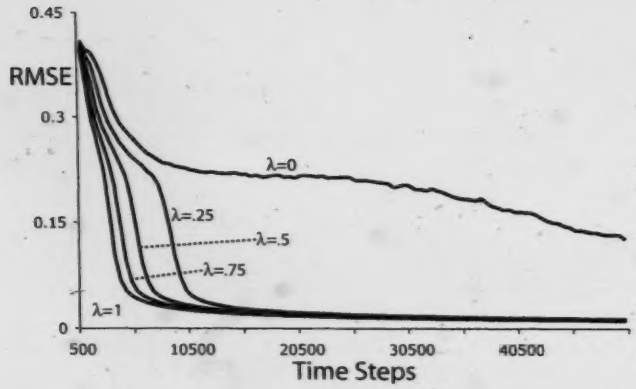

Figure 5.6: Learning curves of our algorithm for various values of λ on the 5-state ring world. This chart represents the average RMSE over all of the nodes in the TD(λ) network. The question network used is of the form seen in Figure 5.1, a full, symmetric, action-conditional question network with depth $d = 3$. Each data point in this graph is the average error of the network over 500 time steps. These results are the average of 50 trials.

increasing λ reduced the amount of data that was required to reach a the same error level. In the extreme TD(0) case, the model will not reach RMSE $< .05$ until over 150 000 time steps have passed.

The number of nodes in fully symmetric question networks rises exponentially with the depth of the network, making it quite costly to make longer predictions. There is a smaller question network that can represent the ring world shown in Figure 5.5. The size of this question network scales linearly with the number of states in the ring. In this experiment, TD(0) could not find a solution to the 8-state ring world in any of the configurations that were tried. Although not shown, the experiment continued for over ten million steps and the TD(0) networks did not improve. TD($\lambda > 0$) was able to solve this problem for all values of λ that were tried. Again, increasing λ decreases the amount of data required by the algorithm.

Although not presented in detail here, we have seen similar improvements with TD(λ) vs TD(0) on other problems such as the partially-observable random walk

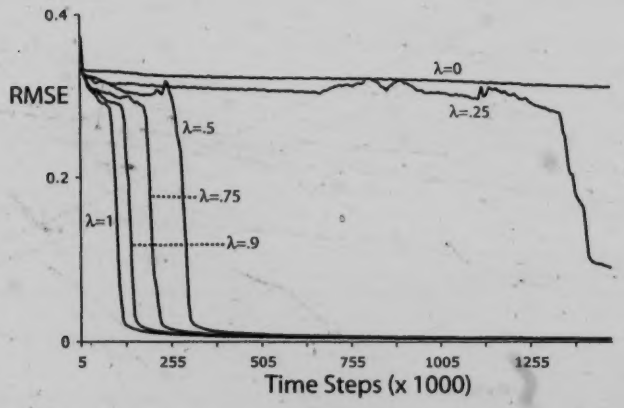

Figure 5.7: Learning curves of our algorithm for various values of λ on the 8-state ring world. The question network used is of the form in Figure 5.5 with depth $d = 8$. This chart represents the average RMSE over all of the nodes in the TD(λ) network. Each data point in this graph is the average error of the network over 5000 time steps. These results are the average of 50 trials.

world and Littman *et al.*'s float-reset problem [2002].

Complexity of the Cycle and Ring Worlds

The domains used in these experiments behave deterministically but have extreme state aliasing. It is conceivable that these environments are trivial, and that our success is not encouraging. To test this theory, we have attempted to learn POMDP models of these three environments using the EM (Baum-Welch) algorithm.[2]

	RMSE (amount of data)	
Problem	EM	TD(1)
6 State Cycle	.313 (10 000)	.05 (5 000)
5 State Ring	.37 (10 000)	.05 (5 000)
8 State Ring	.28 (250 000)	.05 (125 000)

Table 5.1: RMSE of EM and TD(1) algorithm on the 6-state cycle, 5-state ring, and 8-state ring worlds. Error is calculated by comparing the learned POMDP observation predictions to the true probabilities over a 1000 step test sequence.

[2]Code graciously provided for us for comparison was the same as used in Wolfe *et al.* [2005].

For each domain, we provided EM with the correct number of nominal states and ran 100 trials of 20 iterations each. In each case the EM algorithm was provided with more data than the TD(1) learning algorithm required. In Table 5.1 the minimum RMSE error achieved of any of the 100 trials with the EM algorithm is compared to our TD(1) results. The TD network model does considerably better than the model learned with EM in all cases. The results strongly suggest that the cycle and ring worlds are not trivial. Very little time was spent tuning the parameters of the EM algorithm; these results are not meant in any way to suggest TD networks are superior or inferior to learning POMDPs with EM.

5.4 The Computational Cost of λ

The benefits that are gained by using TD(λ) over TD(0) must come at a cost. In our algorithm, memory and computation resource usage grows at approximately the same rate, collectively they are called the *cost*. We consider two different families of question network that exemplify the additional cost of TD(λ) networks over TD(0) networks.

First, consider an unconditioned question network like the one shown in Figure 4.1, but of an arbitrary depth, *d*. This is the *chain* question network. Second, consider an action-conditional question network like that in Figure 5.1, but with arbitrary branching factor (number of actions) b and depth d. This is the *tree* question network.

The largest computational cost of either algorithm is the number of weight updates that are performed at each time step. Other book-keeping costs are negligible and are not included in the analysis.

With an unconditional "chain" question network, the number of updates for TD(0) at each step will be one update for each node, or equal to the depth of the network, d. In a TD(λ) network of the same depth, the number of updates is:

$$\text{Chain network updates for TD}(\lambda) = \frac{d(d+1)}{2}$$

The ratio of the cost of TD(λ) over TD(0) gives us a measure of the factor of additional cost of TD(λ). This additional cost factor is:

$$\text{Chain network work ratio} = \frac{d+1}{2}$$

This is an upper bound on the additional cost that will be incurred by TD(λ) networks for any *single-target* question network. This is a degenerate case, where the length of the longest question is equal to the number of nodes in the network, and every prediction always has a target. In practice, TD networks will ask a variety of action-conditional questions of different lengths, and they will not always have a valid target. For this reason, our primary interest is not the "chain" network, it is the "tree" question network.

In a tree question network with depth d and branching factor (number of actions) b , the number of updates on each time step for the TD(0) learning algorithm is:

$$\text{Tree network updates for TD}(0) = \frac{b^d - 1}{b - 1}$$

For the same network, the number of updates performed by the TD(λ) learning

algorithm is:

$$\text{Tree network updates for TD}(\lambda) = \frac{b^{d+1} - (d+1)b + d}{(b-1)^2}$$

The ratio of these two costs can be used to make a rough estimate of the extra work required to use TD(λ) for a given question network. This ratio is:

$$\text{Tree network work ratio} = \frac{b^{d+1} - (d+1)b + d}{(b-1)(b^d - 1)}$$

This equation holds as long as $b > 1$ and $d >= 1$. When $b = 2$, this ratio rises from 1.0 to a maximum of 2.0 as the depth of the network increases. As the number of actions increase, this ratio decreases to $1.0 + \epsilon$, a negligible amount of extra work.

In future applications of TD(λ) networks a variety of question networks will be used. The topology of these networks (on average) will fall somewhere between the chain and tree question networks. When the question network is not a full tree, the number of node updates per step will depend on the policy being followed. Some of the ring world experiments in this chapter used a question network with $d = 8$ which had only 16 nodes. In this case, the formulae predict that for $d = 8$, the number of node updates will be 16 for a symmetric tree and 36 for a chain network. The average number of node updates per step for this question network is 14.5, less than either estimate.

We propose one modification to our algorithm that would mitigate the additional cost of TD(λ), if required. In our algorithm, the target v_t^i is based on the full parental hierarchy of the prediction, right up to the observation bit. One simple change to the algorithm could cut off these traces after they have used some bounded number

of targets. This "bounded lookahead" parameter would allow the number of updates per step to be tuned between d and $\frac{d(d+1)}{2}$ easily to suit any particular situation.

5.5 Conclusions

In all of our experiments, TD(λ) networks with $\lambda > 0$ have learned faster than with $\lambda = 0$. This is strong evidence that our generalized TD(λ) network learning algorithm is an improvement over the existing TD(0) learning algorithm. TD(λ) networks have also solved problems that were not solvable with the TD(0) learning algorithm. These problems may be solvable because the multi-step backups of the TD(λ) learning algorithm eliminate information flow dependencies between the question and answer networks. The cost of the TD(λ) network learning algorithm is less than twice that of the TD(0) algorithm for the types of questions that are important to represent a model of a controlled dynamical system (action-conditional questions). For some other question networks (the chain network) the additional cost is larger, but with simple techniques such as adding a bounded lookahead parameter, this cost can easily be controlled.

In the conventional TD(λ) learning algorithm, no single value of λ is always best. It is surprising that our experiments suggest $\lambda = 1$ is better than any other value for λ. This disparity may be related to our class of problems, partially-observable (non-Markov) environments. In reinforcement learning, the value of λ can be thought of as a parameter to specify a mixture of TD and Monte Carlo backups. TD is more data efficient and requires less computation while Monte Carlo is more robust in non-Markov environments. In Chapter 3, TD(0) backups were better than Monte Carlo backups for TD network learning in certain Markov environments. In this

TD(λ) work, we have only considered partially-observable environments, a situation that favors Monte Carlo or TD(1) backups. It is intuitive that in this case, learning favors higher values of λ.

Chapter 6

Conclusion

6.1 Discussion and Contributions

In this thesis we have introduce temporal-difference networks as a formalism for expressing and learning grounded world knowledge in a predictive form. TD networks suggest a much larger set of possibilities than conventional TD methods for learning to predict. Chapter 3 explored the first few of these possibilities. In a fully observable setting there are sometimes significant advantages to using TD learning when learning TD-defined predictions. The action-conditional results in that chapter shows that TD methods can learn dramatically faster than other methods. TD networks allow the expression of many new kinds of predictions whose extensive semantics is not immediately clear, but which are ultimately fully grounded in data.

TD networks have been extended to incorporate the strengths of history-based methods in order to better learn models of non-Markov environments. This combined approach leverages history information to learn a better predictive representation.

In Chapter 5, the TD(0) updates of the TD network learning algorithm were generalized to create TD(λ) networks. In our experiments, TD(λ) networks with $\lambda > 0$ learn faster than with $\lambda = 0$. This is strong evidence that the generalized TD(λ) network learning algorithm is an improvement over the TD(0) learning algorithm. TD(λ) networks have also solved problems that were not solvable with the TD(0) learning algorithm. These problems may be solvable because the multi-step backups of the TD(λ) learning algorithm help eliminate information flow dependencies between the question and answer networks.

The scope of the work presented in this thesis is large and leaves us with many open questions. TD networks can learn predictive models of some small test environments. The representational power of TD networks is still unknown, there may be environments that cannot be represented by TD networks. It is not clear how our learning algorithms will fare as the size and complexity of these environments increase. It may be difficult to learn a TD network model even if the question network is known to be sufficient. When the question network is not sufficient, there is evidence that TD network learning is unstable, as shown in Chapter 4.1.2.

In the face of these open questions, TD networks are a novel research area that we intend to pursue further.

6.2 Future Work

There are many interesting, open questions about TD networks. The following sections briefly introduce a few open areas that we plan on investigating in our future work.

6.2.1 Discovery

How can we create question networks dynamically to suit a particular task? *Discovery* is the procedure of automatically finding an appropriate question network. The current approach of manually specifying the structure of the question network is only feasible for simple tasks that can either be analytically discovered or represented by a symmetric action conditional tree network of a small depth. It will soon be important to have a method of growing an appropriate question network through experience with the system. We are currently working on a heuristic, online, incremental algorithm that will further reduce the need for designer influence in learning predictive models of dynamical systems.

6.2.2 Fast Learning

The techniques that are currently used to learn predictions in a TD network are not efficient. Although TD networks can learn models from a small batch of data that is presented over and over in an offline fashion, TD networks are not data efficient when operating online. TD network learning is slow for a few reasons. First, the method of using big tree question networks creates redundant nodes in the question network. Redundant questions lead to redundant inputs, which is known to slow learning. A discovery algorithm will help to eliminate this redundancy, which will in turn speed up learning. We are investigating other methods of directly identifying and removing correlation in the input vector. We are also interested in using the incremental delta-bar-delta algorithm to adjust the learning rate on a per-feature basis, so that the learning update puts more weight on the features with the most information content [Sutton, 1992].

6.2.3 Temporal Abstraction

Sutton *et al.* [2005] have suggested that TD networks can be extended to ask temporally abstract questions, using the options framework [Sutton et al., 1999]. Their preliminary results indicate that the options framework may offer a way to scale TD networks to much larger environments.

6.2.4 Reinforcement Learning

Our eventual ambition is that TD network models of an environment could be used by a sequential decision making algorithm to perform tasks in the world. In future work we intend to apply reinforcement learning directly to TD network representations. Rafols et al. [2005] have seen promising results while exploring the general implications of using predictive representation as state representations for reinforcement learning.

Bibliography

[Boyan, 2002] Boyan, J. A. (2002). Least-squares temporal difference learning. *Machine Learning*, 49(2-3):233–246.

[Bradke and Barto, 1996] Bradke, S. J. and Barto, A. G. (1996). Linear least-squares algorithms for temporal difference learning. *Machine Learning*, 22(1-3):33–57.

[Drescher, 1991] Drescher, G. L. (1991). *Made-up minds: a constructivist approach to artificial intelligence*. MIT Press.

[Elman, 1990] Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, 14(2):179–211.

[Holmes and Isbell, Jr., 2005] Holmes, M. P. and Isbell, Jr., C. L. (2005). Schema learning: Experience-based construction of predictive action models. In *Advances in Neural Information Processing Systems 17*, pages 585–592. MIT Press, Cambridge, MA.

[Jaeger, 1998] Jaeger, H. (1998). Discrete-time, discrete-valued observable operator models: a tutorial. Technical report, German National Research Center for Information Technology.

[James and Singh, 2004] James, M. R. and Singh, S. (2004). Learning and discovery of predictive state representations in dynamical systems with reset. In *Proceedings of the Twenty-First International Conference on Machine Learning*, pages 719–726.

[Jordan, 1986] Jordan, M. I. (1986). *The learning of representations for sequential performance*. PhD thesis, University of California in San Diego.

[Lagoudakis and Parr, 2003] Lagoudakis, M. and Parr, R. (2003). Least-squares policy iteration. *Journal of Machine Learning Research*, 4:1107–1149.

[Littman, 1996] Littman, M. L. (1996). *Algorithms for sequential decision-making.* PhD thesis, Brown University.

[Littman et al., 2002] Littman, M. L., Sutton, R. S., and Singh, S. (2002). Predictive representations of state. In *Advances in Neural Information Processing Systems 14*, Cambridge, MA. MIT Press.

[McCallum, 1995] McCallum, A. (1995). Instance-based utile distinctions for reinforcement learning with hidden state. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 387–395.

[Mehrotra et al., 1997] Mehrotra, K., Mohan, C. K., and Ranka, S. (1997). *Elements of Artificial Neural Networks*, chapter 4, pages 136–139. MIT Press.

[Pineda, 1988] Pineda, F. J. (1988). Generalization of backpropagation to recurrent and higher order neural networks. In *Advances in Neural Information Processing Systems 1*, pages 602–611.

[Rafols et al., 2005] Rafols, E. J., Ring, M. B., Sutton, R. S., and Tanner, B. (2005). Using predictive representations to improve generalization in reinforcement learning. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, pages 835–840.

[Ring, 1994] Ring, M. B. (1994). *Continual Learning in Reinforcement Environments.* PhD thesis, University of Texas at Austin, Texas.

[Rivest and Schapire, 1990] Rivest, R. L. and Schapire, R. E. (1990). A new approach to unsupervised learning in deterministic environments. In *Machine Learning, An Artificial Intelligence Approach*, volume III, pages 670–684. Morgan Kaufmann Publishers Inc.

[Rivest and Schapire, 1994] Rivest, R. L. and Schapire, R. E. (1994). Diversity-based inference of finite automata. *Journal of the Association for Computing Machinery*, 41(3):555–589.

[Rudary and Singh, 2004] Rudary, M. R. and Singh, S. (2004). A nonlinear predictive state representation. In *Advances in Neural Information Processing Systems 16*, Cambridge, MA. MIT Press.

[Rumelhart et al., 1986] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by error propagation. In *Parallel Distributed Processing.*

[Russell and Norvig, 2003] Russell, S. J. and Norvig, P. (2003). *Artificial Intelligence: A Modern Approach.* Pearson Education.

[Shalizi and Shalizi, 2004] Shalizi, C. R. and Shalizi, K. L. (2004). Blind construction of optimal nonlinear recursive predictors for discrete sequences. In *Proceedings of the Twentieth Conference on Uncertainty in Artificial Intelligence*, pages 504–511.

[Shen, 1993] Shen, W. M. (1993). Learning finite state automata using local distinguishing experiments. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 1088–1093.

[Singh et al., 2004] Singh, S., James, M. R., and Rudary, M. R. (2004). Predictive state representations: A new theory for modeling dynamical systems. In *Proceedings of the Twenthieth Conference on Uncertainty in Artificial Intelligence*, pages 512–519.

[Singh et al., 2003] Singh, S., Littman, M. L., Jong, N., Pardoe, D., and Stone, P. (2003). Learning predictive state representations. In *Proceedings of the Twentieth International Conference on Machine Learning*, pages 712–719.

[Sutton, 1988] Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44.

[Sutton, 1992] Sutton, R. S. (1992). Adapting bias by gradient descent: An incremental version of delta-bar-delta. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, pages 171–176, San Jose, CA.

[Sutton and Barto, 1998] Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning : An Introduction*. MIT Press.

[Sutton et al., 1999] Sutton, R. S., Precup, D., and Singh, S. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112:181–211.

[Sutton et al., 2005] Sutton, R. S., Rafols, E. J., and Koop, A. (2005). Temporal abstraction in TD networks. Technical report, University of Alberta.

[Sutton and Tanner, 2005] Sutton, R. S. and Tanner, B. (2005). Temporal-difference networks. In *Advances in Neural Information Processing Systems 17*, pages 1377–1384, Cambridge, MA. MIT Press.

[Tanner and Sutton, 2005a] Tanner, B. and Sutton, R. S. (2005a). Temporal-difference networks with eligibility traces : TD(λ) networks. In *Proceedings of the 22nd International Conference on Machine Learning*, pages 889–896.

[Tanner and Sutton, 2005b] Tanner, B. and Sutton, R. S. (2005b). Temporal-difference networks with history. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, pages 865–870.

[Williams and Zipser, 1989] Williams, R. J. and Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1:270–280.

[Wolfe et al., 2005] Wolfe, B., James, M. R., and Singh, S. (2005). Learning predictive state representations in dynamical systems without reset. In *Proceedings of the 22nd International Conference on Machine Learning*, pages 985–992.